Árvore AVL

Prof. Jose J. Camata
Prof. Marcelo Caniato
Prof. Barbara Quintela
camata@ice.ufjf.br
marcelo.caniato@ice.ufjf.br
barbara@ice.ufjf.br

ufjf

Tópicos

1. Introdução
2. Conceito de balanceamento
3. Árvore AVL
4. Rotações
5. Exemplos

Introdução

- Árvores são estruturas interessantes por dois motivos
  - Relação de hierarquia
  - **Velocidade na busca**

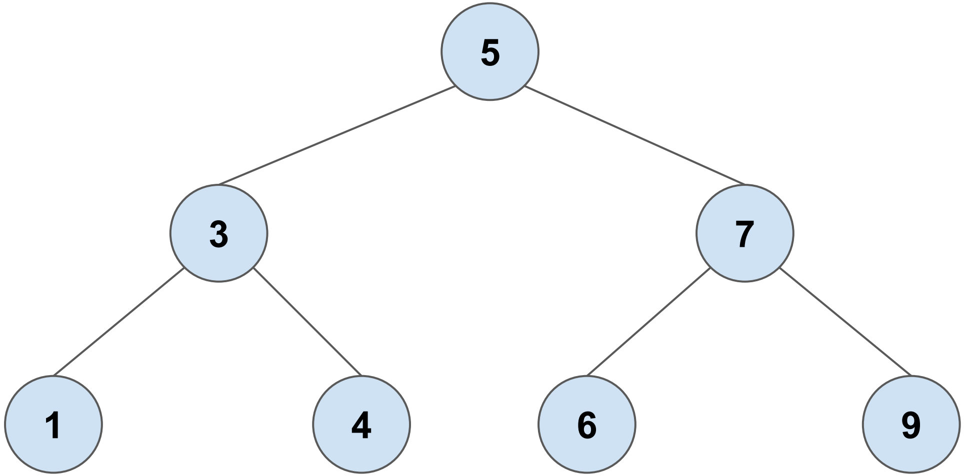
Para árvores criadas a partir de uma entrada de dados aleatória, a busca é O(logn)

Podemos garantir O(logn) em todos os casos?

Introdução

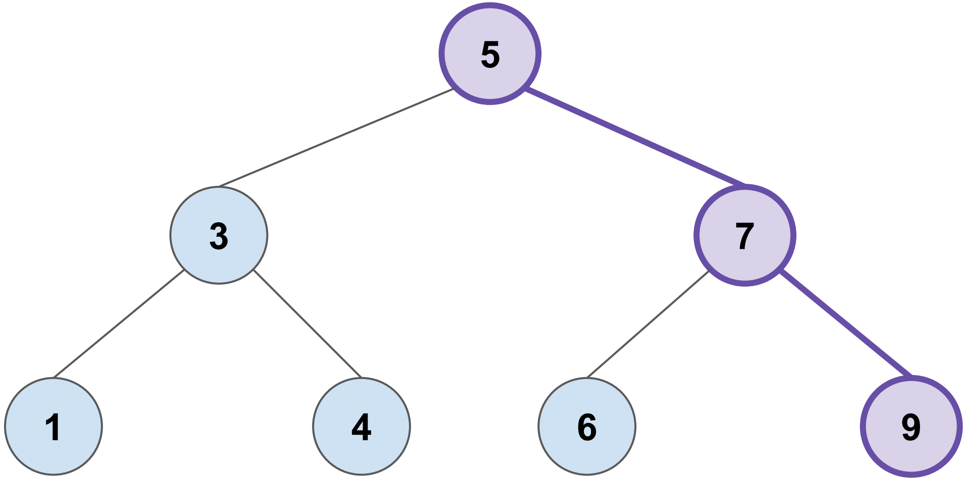
- O pior caso é a busca não sucedida, que é O(h), em que h é a altura da árvore
  - Exemplo: buscar 8 na árvore abaixo

Ordem de inserção:
5 3 1 4 7 6 9



Introdução

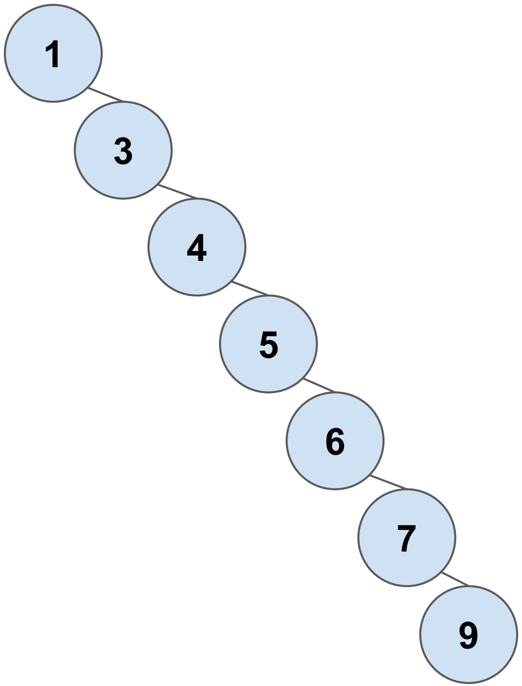
- O pior caso é a busca não sucedida, que é O(h), em que h é a altura da árvore
  - Exemplo: buscar 8 na árvore abaixo



Apenas 3 comparações

Introdução

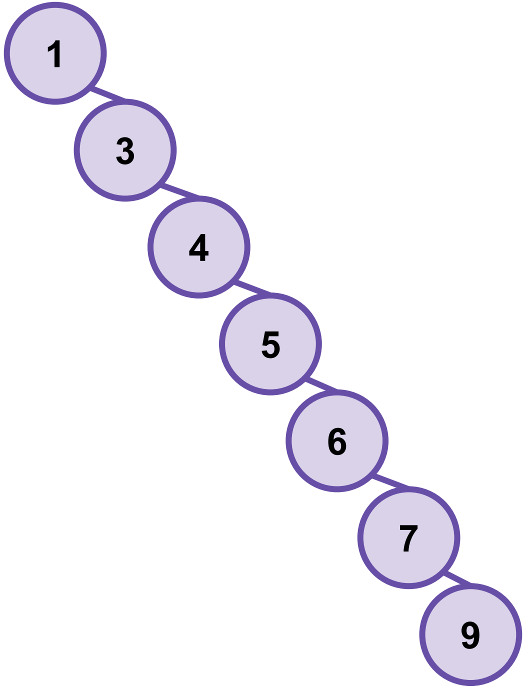
➢ E se a árvore tiver esta forma?



Ordem de inserção:
1 3 4 5 6 7 9

Introdução

- E se a árvore tiver esta forma?



Todos os nós da árvore foram visitados!

Como podemos evitar este caso?

Balanceamento

- A ideia é manter a altura da árvore em O(logn)
  - Uma árvore que mantém essa propriedade para todas as suas subárvores é chamada de **árvore balanceada**
- Vimos que o desbalanceamento surge da ordem de entrada dos dados
  - Deseja-se balancear a árvore à medida que novos valores são inseridos
    - Precisa ser eficiente

Balanceamento

- Possibilidades de balanceamento

Balanceamento global

- Usar um vetor auxiliar
  - Realiza um percurso in-ordem
  - Armazena no vetor
  - Reinsere na árvore via busca binária
- Algoritmo DSW

Balanceamento local

- Aplicar rotinas de balanceamento após inserções e remoções
- Exs.: AVL, vermelho-preto etc.

Iremos focar nestas estruturas

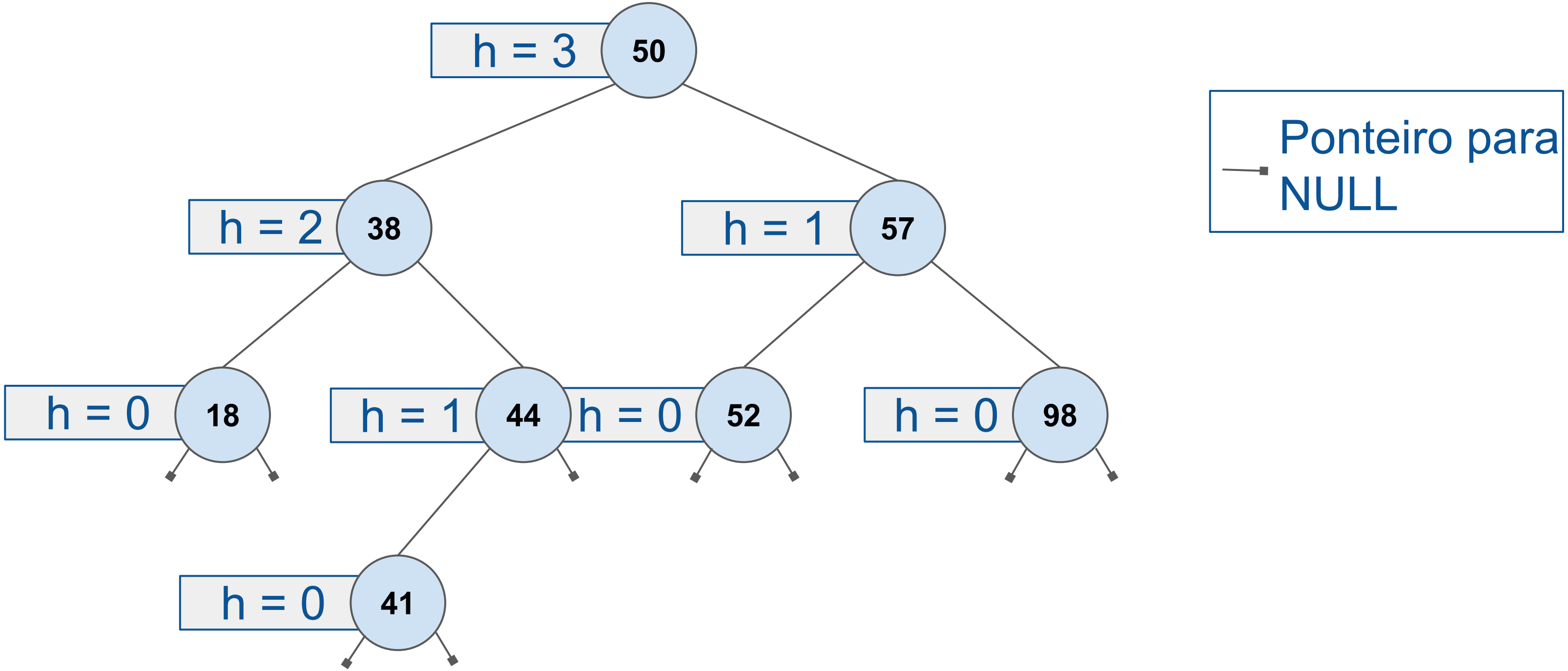
Árvore AVL

- Criada por Adelson-Velskii e Landis
- A diferença de altura entre as subárvores de qualquer nó é de no máximo 1 (+1 ou -1)
- Cada nó possui um fator de balanceamento, dado por:

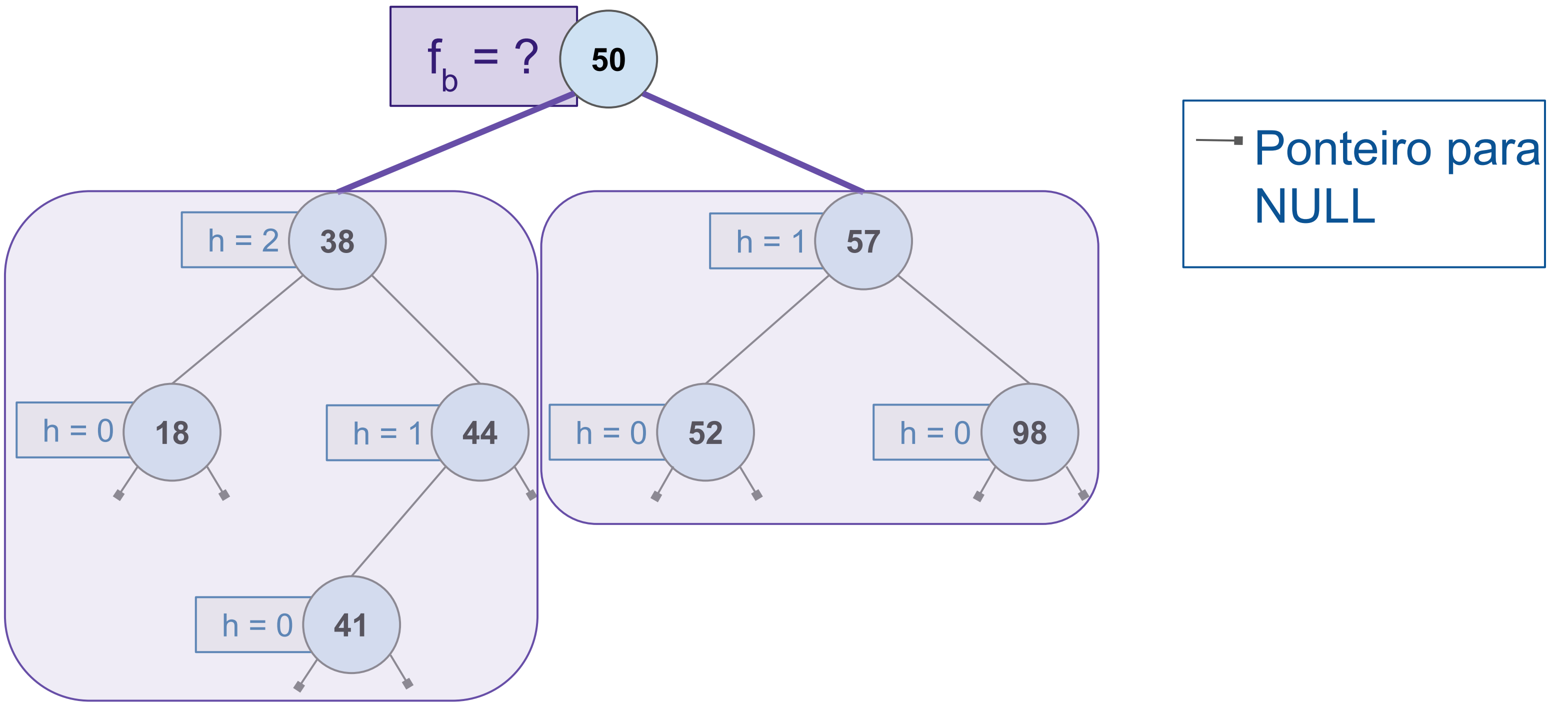
$$f_b = h_d - h_e$$

  - Rebalanceamento realizado sempre que um nó fica com fator +2 ou -2
    - Isto ocorre após uma inserção ou remoção
    - Correção feita através de rotações nas subárvores afetadas

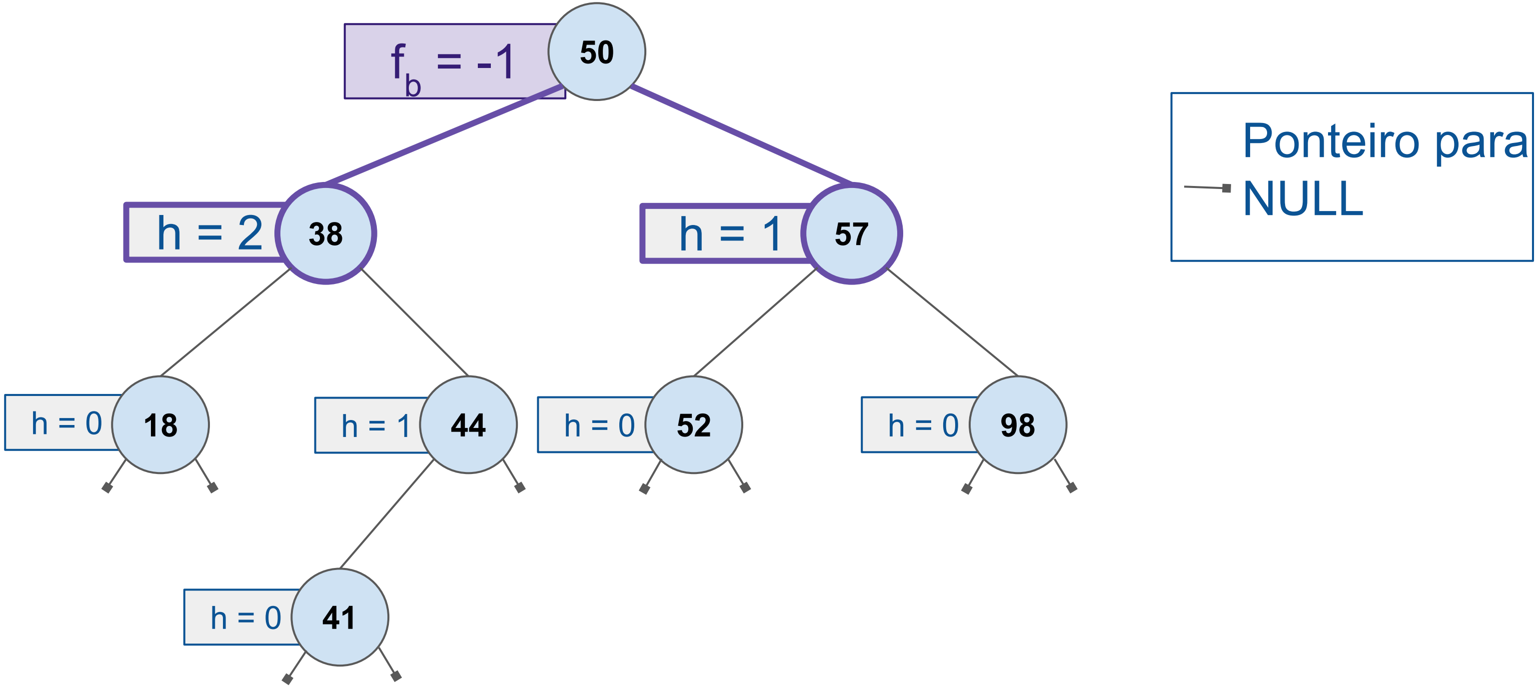
Árvore AVL



Árvore AVL

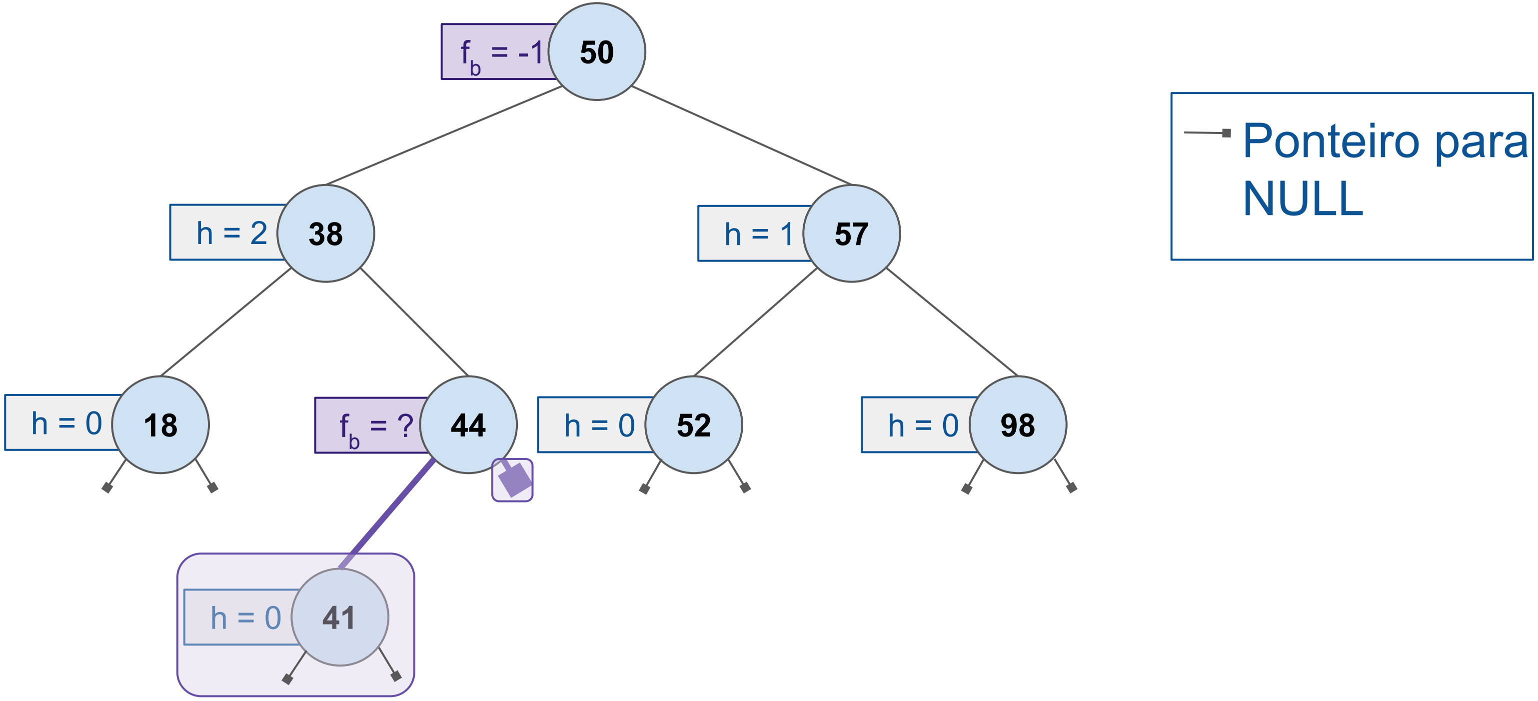


Árvore AVL

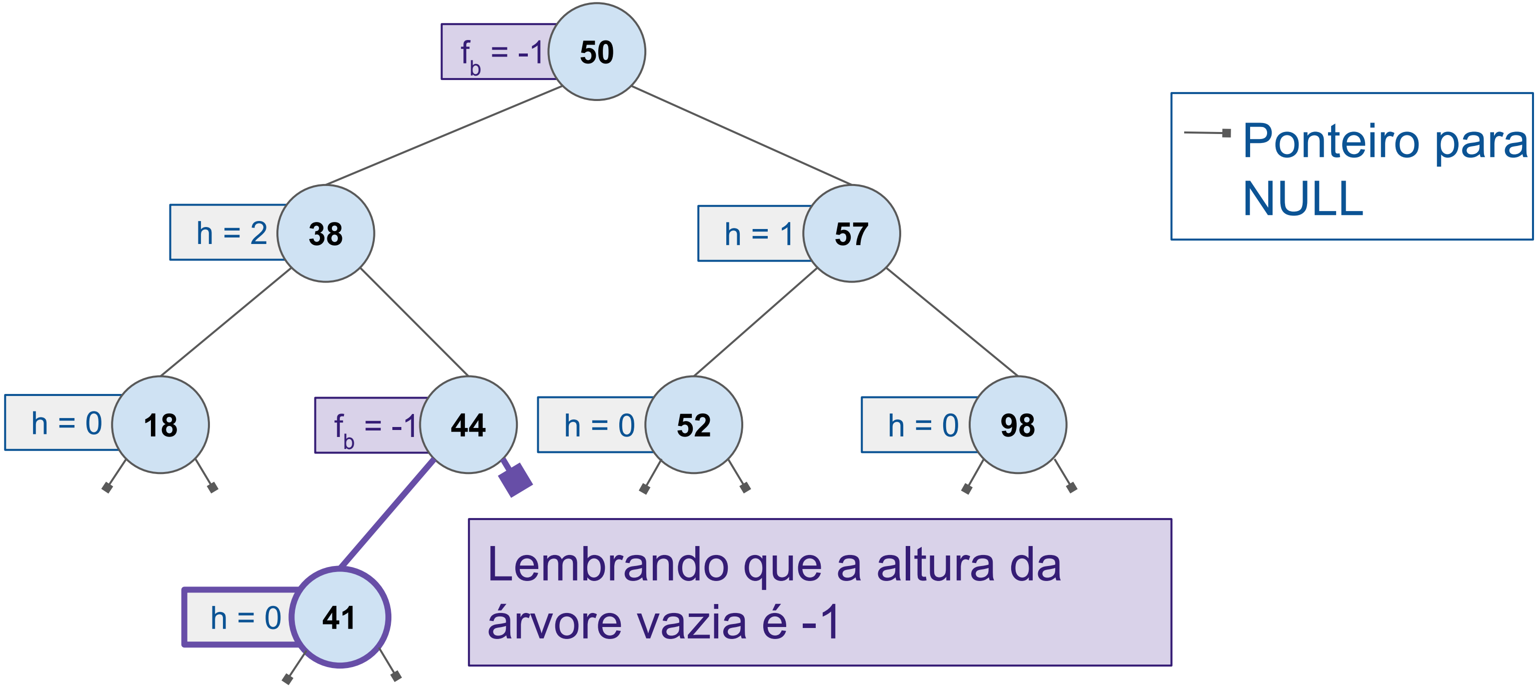


$$f_b = h_d - h_e = 1 - 2 = -1$$

Árvore AVL

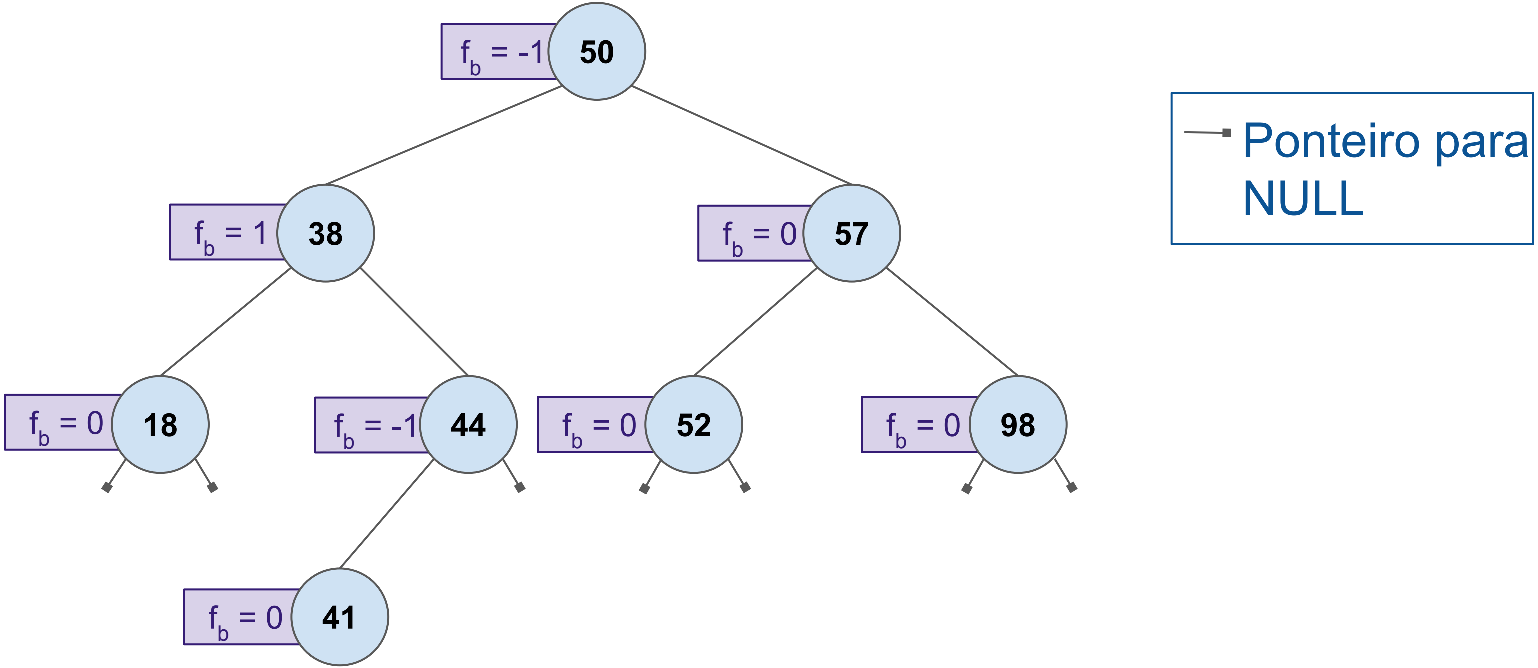


Árvore AVL



$$f_b = h_d - h_e = -1 - 0 = -1$$

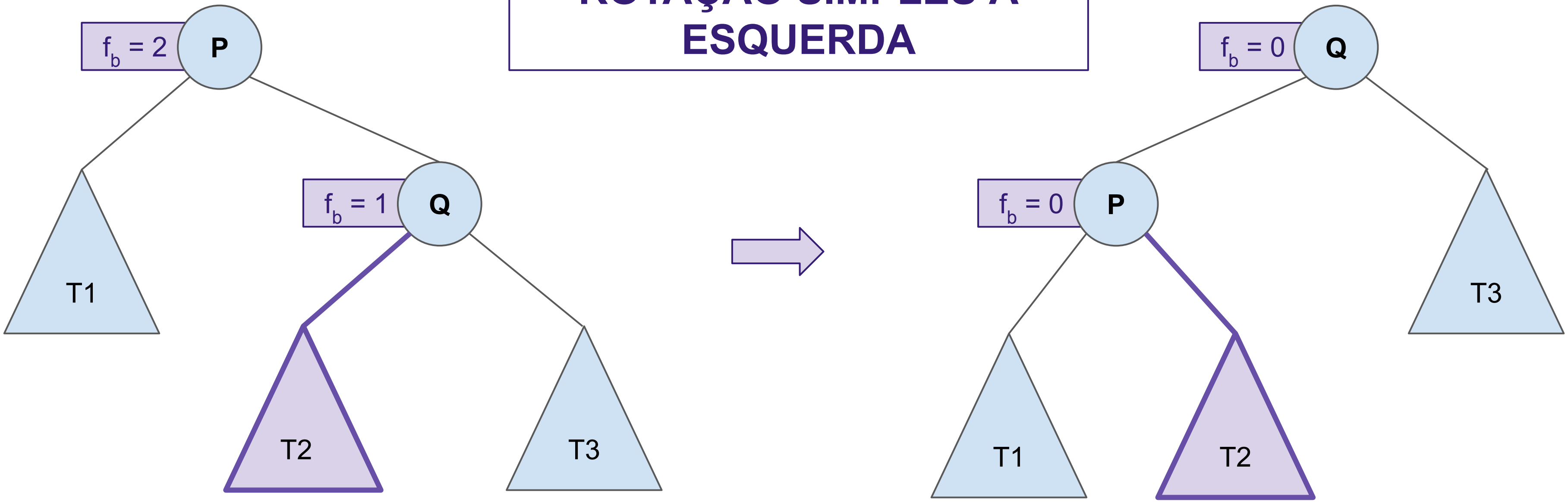
Árvore AVL



f_b = -1
50
f_b = 1
38
f_b = 0
57
f_b = 0
18
f_b = -1
44
f_b = 0
52
f_b = 0
98
f_b = 0
41
Ponteiro para NULL

Rotações

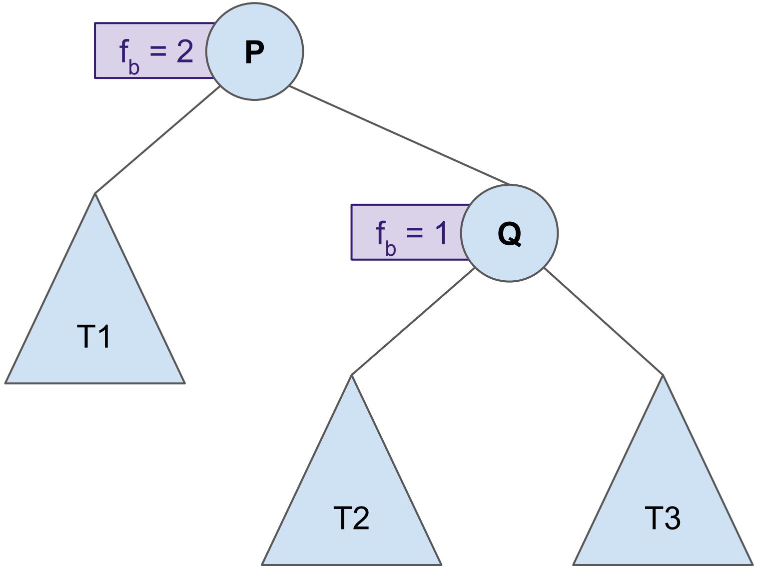
ROTAÇÃO SIMPLES À ESQUERDA



Rotações

ROTAÇÃO SIMPLES À ESQUERDA

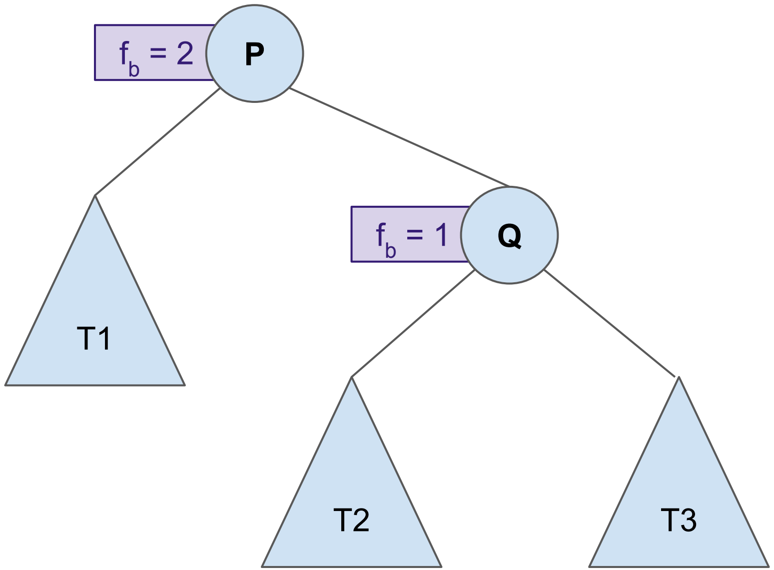
```
Entrada: nó P
1.  q = p->dir
2.  p->dir = q->esq
3.  q->esq = p
```



Rotações

ROTAÇÃO SIMPLES À ESQUERDA

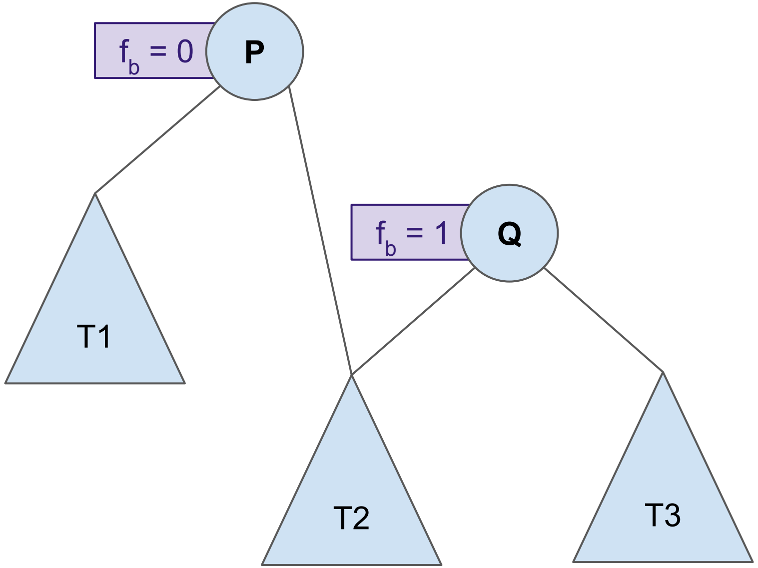
```
Entrada: nó P
1.   q = p->dir
2.   p->dir = q->esq
3.   q->esq = p
```



Rotações

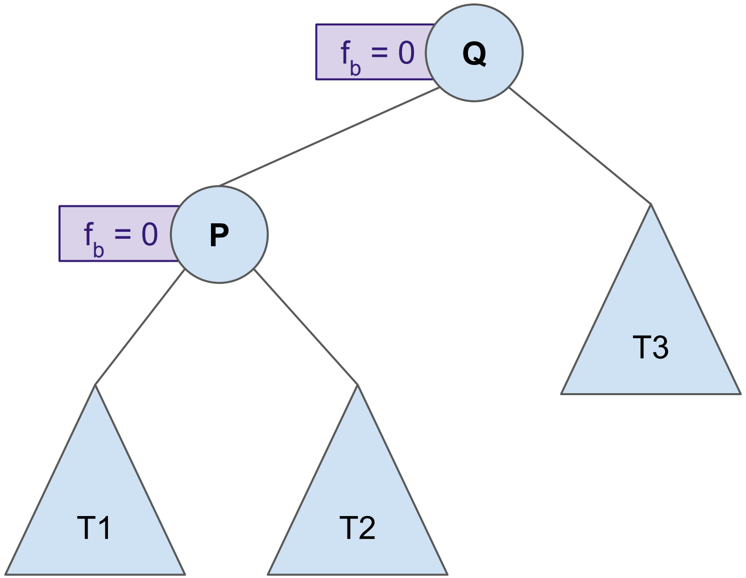
ROTAÇÃO SIMPLES À ESQUERDA

```
Entrada: nó P
1.   q = p->dir
2.   p->dir = q->esq
3.   q->esq = p
```



Rotações

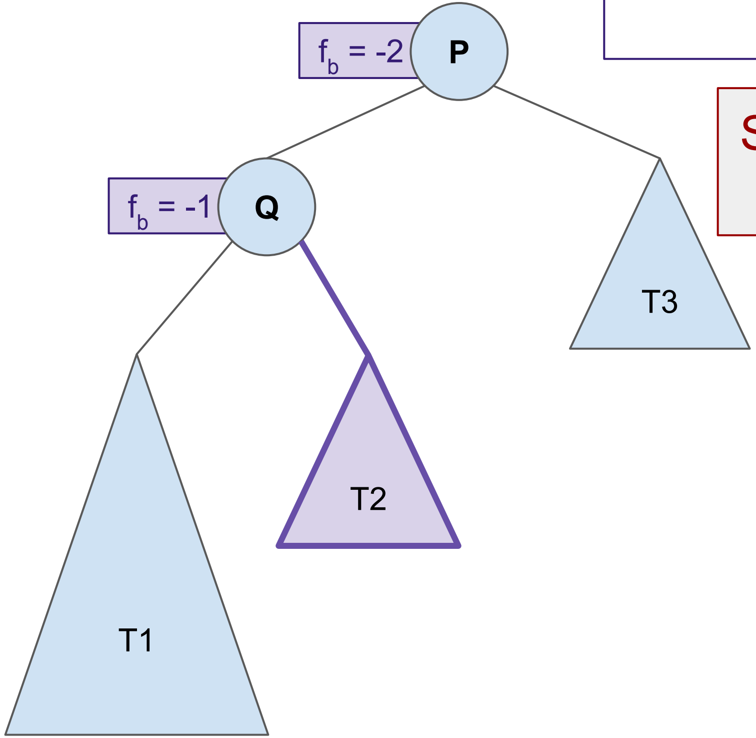
ROTAÇÃO SIMPLES À ESQUERDA
Entrada: nó P 1. q = p->dir 2. p->dir = q->esq 3. q->esq = p

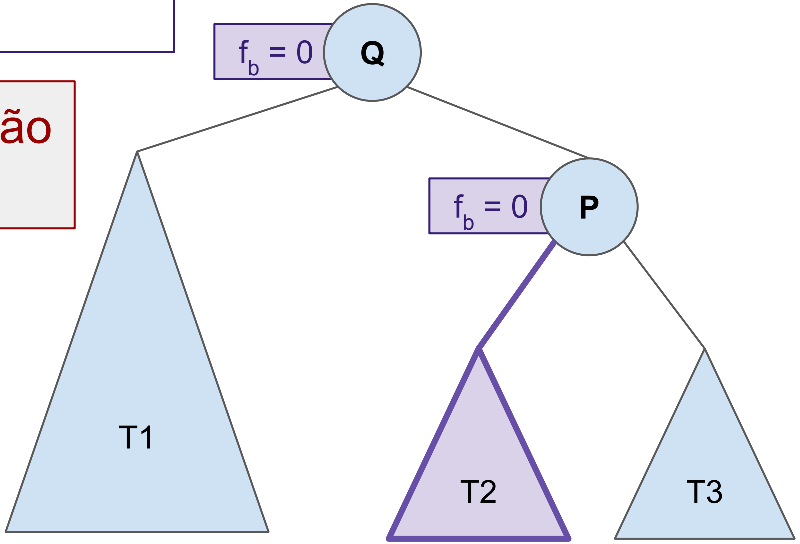


Rotações

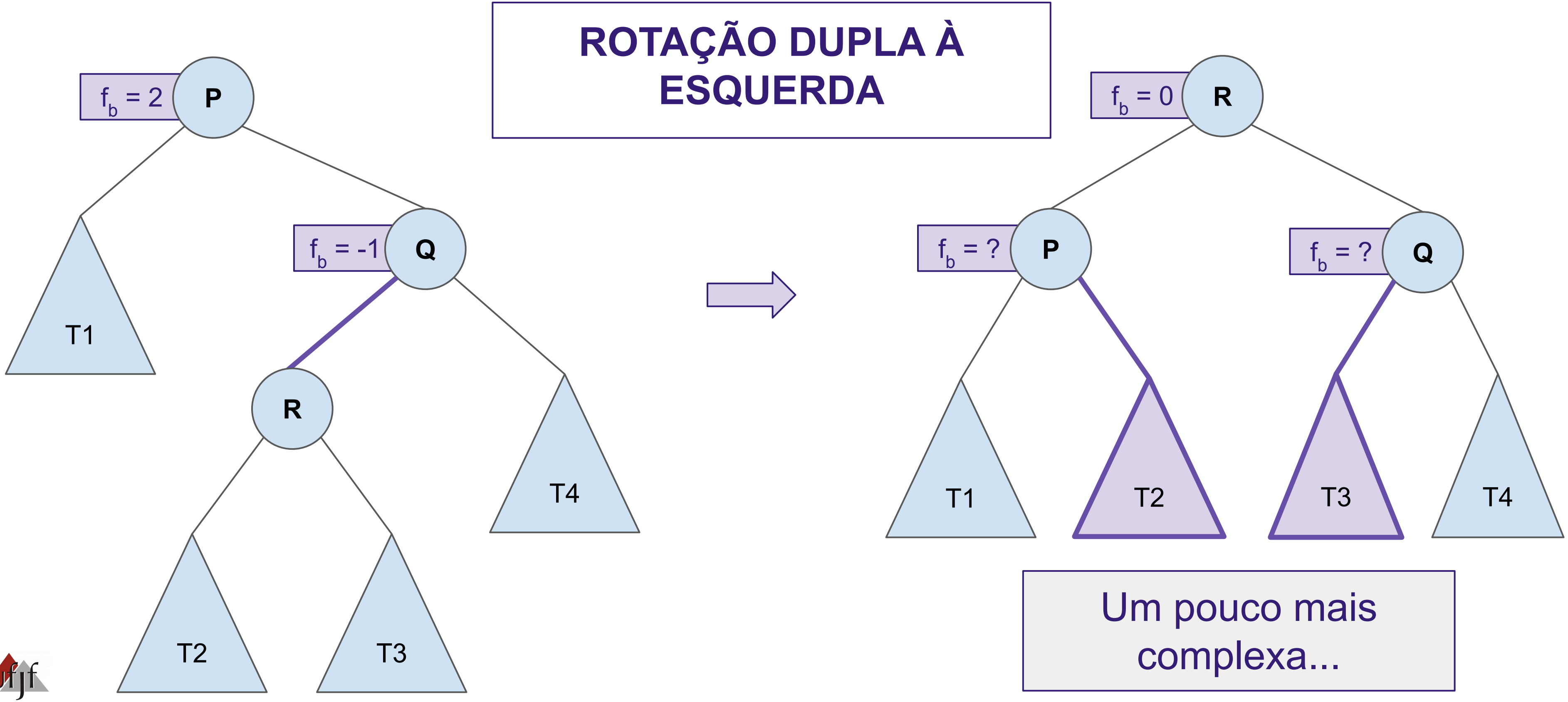
ROTAÇÃO SIMPLES À DIREITA

Simétrico à rotação à esquerda





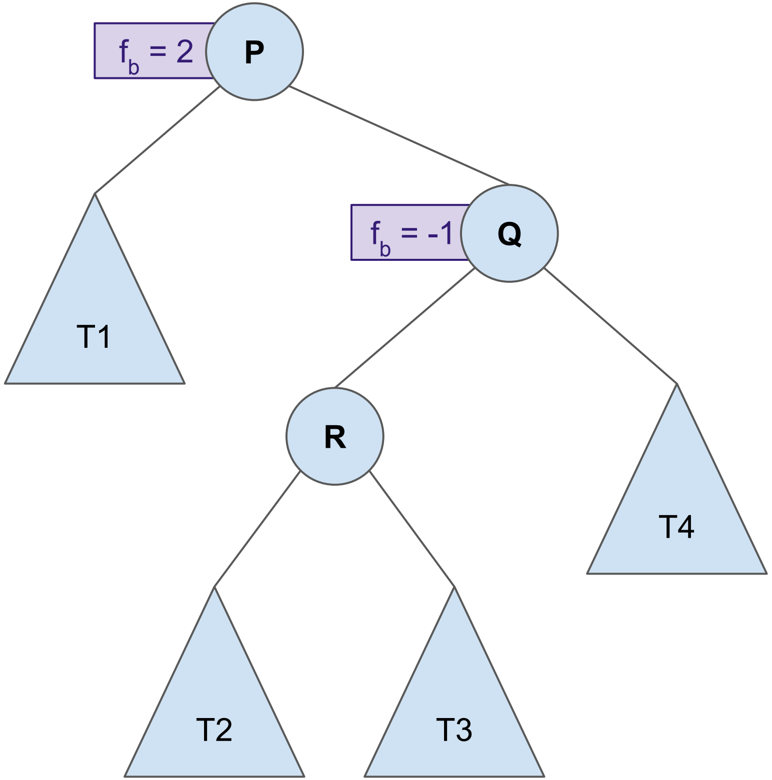
Rotações


ROTAÇÃO DUPLA À ESQUERDA
f_b = 2
P
f_b = -1
Q
R
T1
T2
T3
T4
f_b = 0
R
f_b = ?
P
f_b = ?
Q
T1
T2
T3
T4
Um pouco mais complexa...

Rotações

ROTAÇÃO DUPLA À ESQUERDA

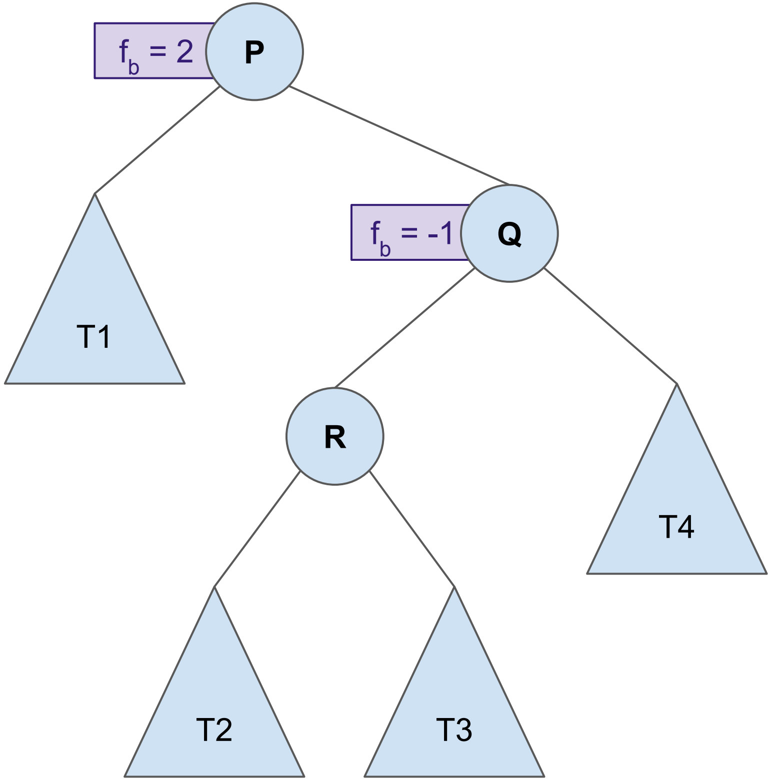
```
Entrada: nó P
1.  q = p->dir
2.  r = q->esq
3.  p->dir = r->esq
4.  q->esq = r->dir
5.  r->esq = p
6.  r->dir = q
```



Rotações

ROTAÇÃO DUPLA À ESQUERDA

```
Entrada: nó P
1.   q = p->dir
2.   r = q->esq
3.   p->dir = r->esq
4.   q->esq = r->dir
5.   r->esq = p
6.   r->dir = q
```

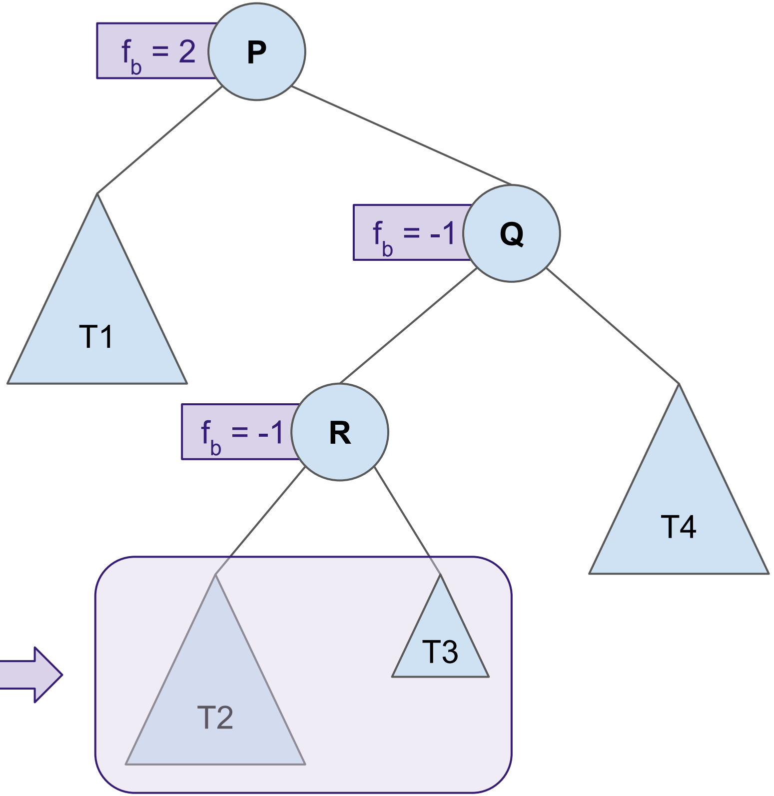


Rotações

ROTAÇÃO DUPLA À ESQUERDA

```
Entrada: nó P
1.   q = p->dir
2.   r = q->esq
3.   p->dir = r->esq
4.   q->esq = r->dir
5.   r->esq = p
6.   r->dir = q
```

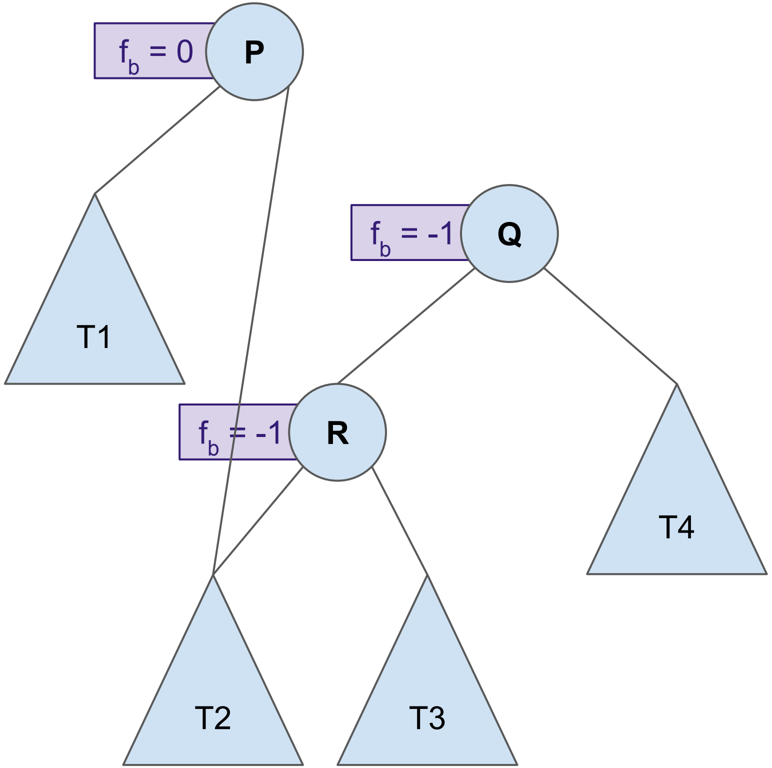
Poderíamos ter também uma configuração invertida aqui (h_e = h-1 e h_d = h)



Rotações

ROTAÇÃO DUPLA À ESQUERDA

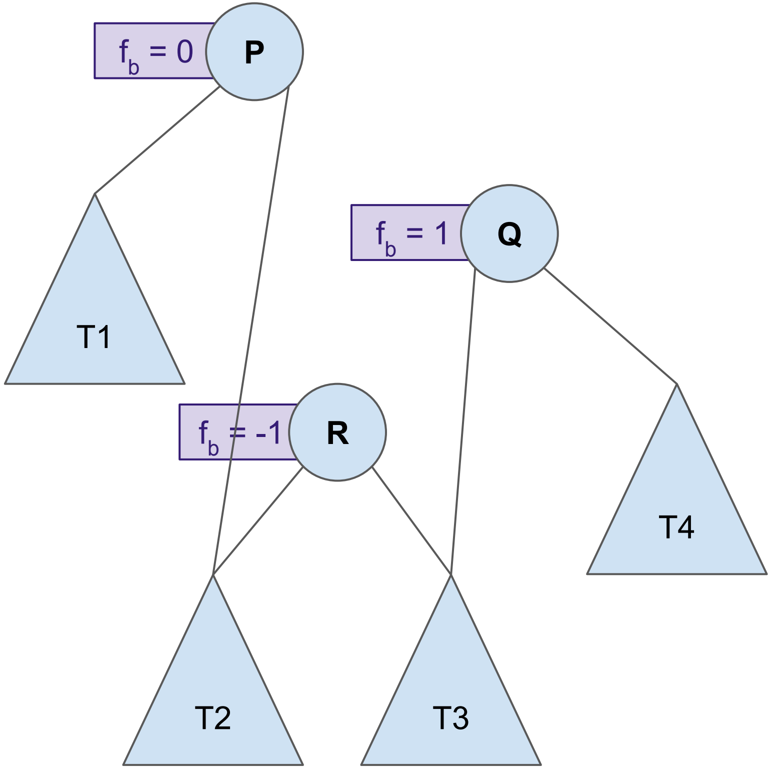
```
Entrada: nó P
1.   q = p->dir
2.   r = q->esq
3.   p->dir = r->esq
4.   q->esq = r->dir
5.   r->esq = p
6.   r->dir = q
```



Rotações

ROTAÇÃO DUPLA À ESQUERDA
Entrada: nó P

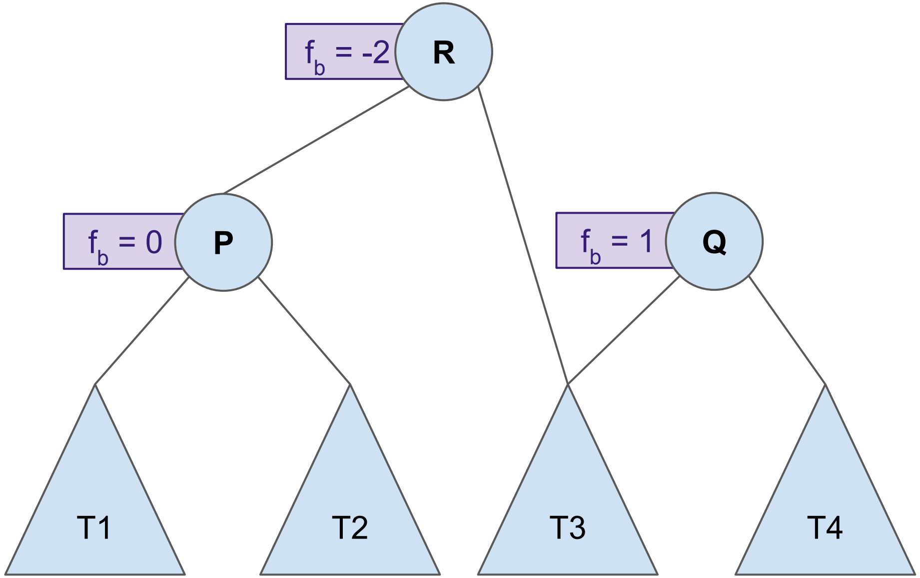
```
1.   q = p->dir
2.   r = q->esq
3.   p->dir = r->esq
4.   q->esq = r->dir
5.   r->esq = p
6.   r->dir = q
```



Rotações

ROTAÇÃO DUPLA À ESQUERDA

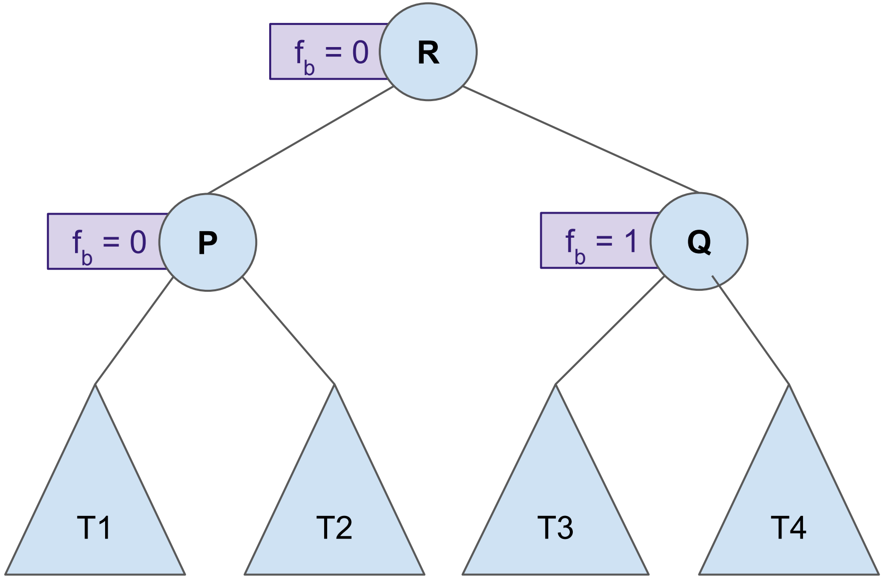
```
Entrada: nó P
1.   q = p->dir
2.   r = q->esq
3.   p->dir = r->esq
4.   q->esq = r->dir
5.   r->esq = p
6.   r->dir = q
```



Rotações

ROTAÇÃO DUPLA À ESQUERDA

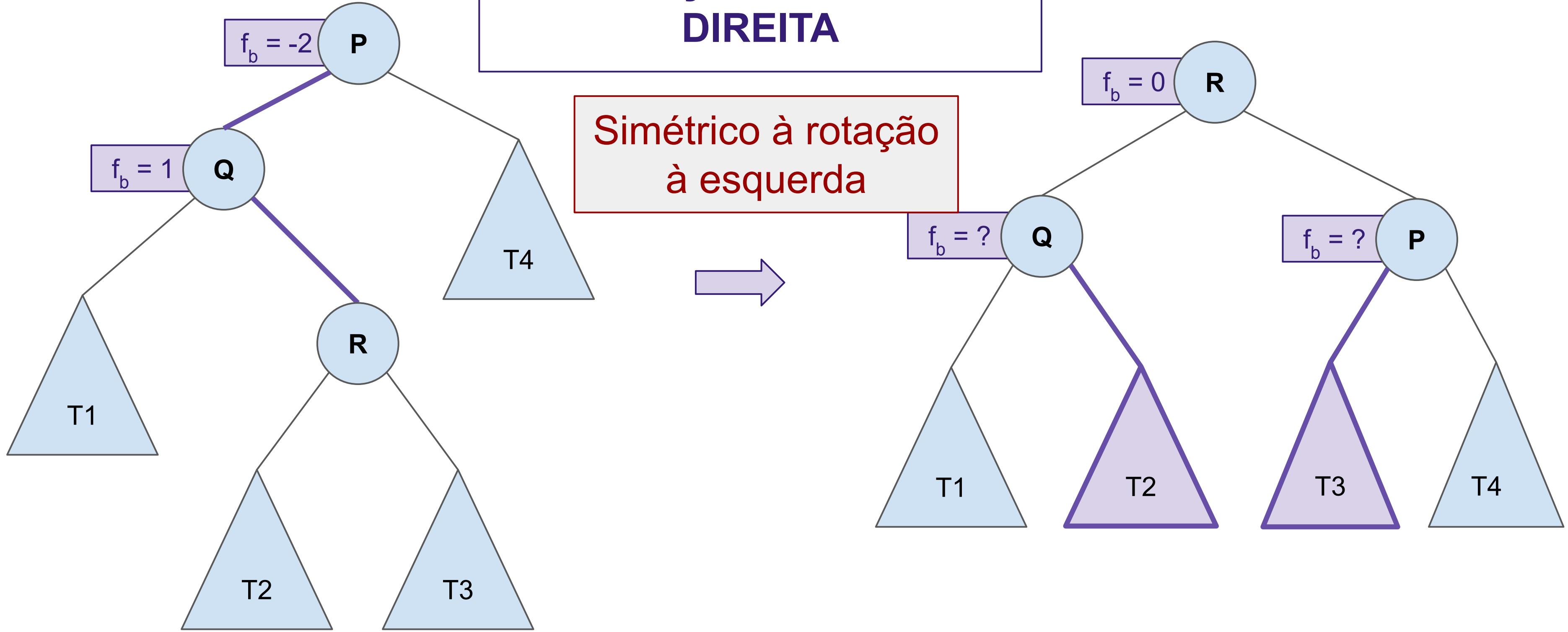
```
Entrada: nó P
1.   q = p->dir
2.   r = q->esq
3.   p->dir = r->esq
4.   q->esq = r->dir
5.   r->esq = p
6.   r->dir = q
```



Rotações

ROTAÇÃO DUPLA À DIREITA

Simétrico à rotação à esquerda

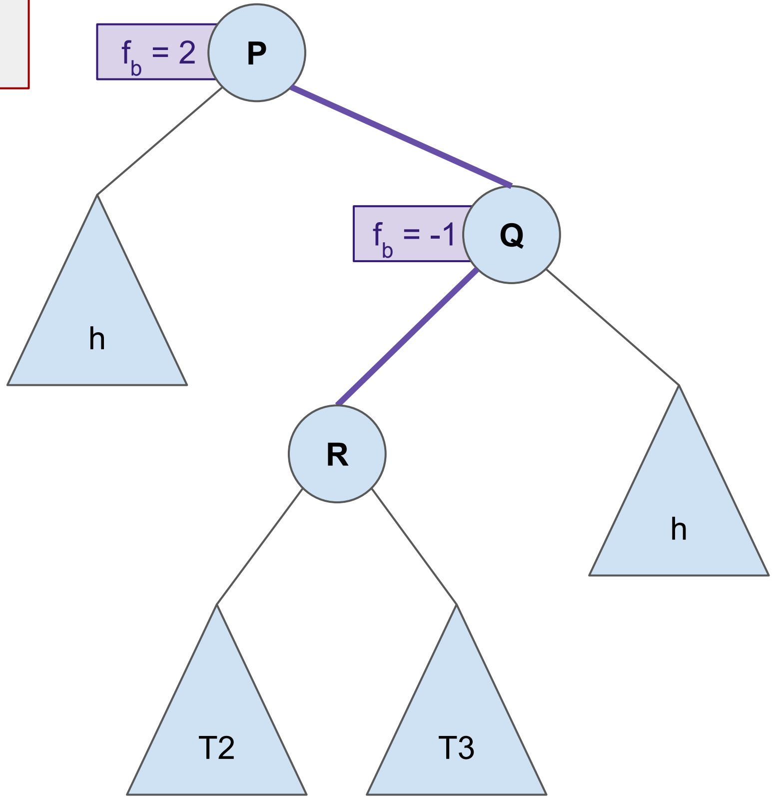


Rotações

A rotação dupla também pode ser vista como uma combinação de duas simples

ROTAÇÃO DUPLA À ESQUERDA

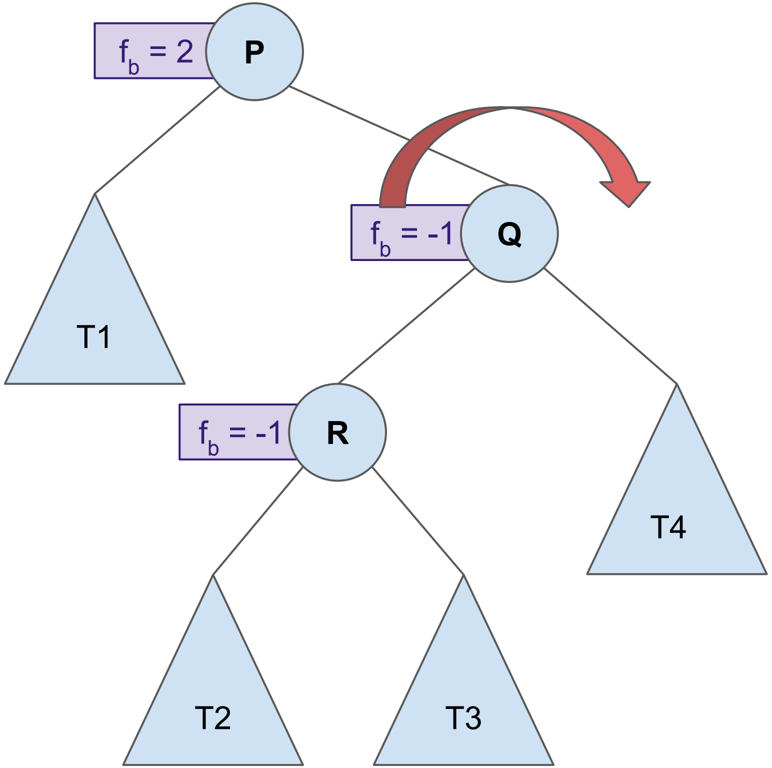
```
Entrada: nó P
1.  rotSimplesDir(q)
  1.1.  r = q->esq
  1.2.  q->esq = r->dir
  1.3.  r->dir = q
2.  rotSimplesEsq(p)
  2.1.  r = p->dir
  2.2.  p->dir = r->esq
  2.3.  r->esq = p
```



Rotações

ROTAÇÃO DUPLA À ESQUERDA

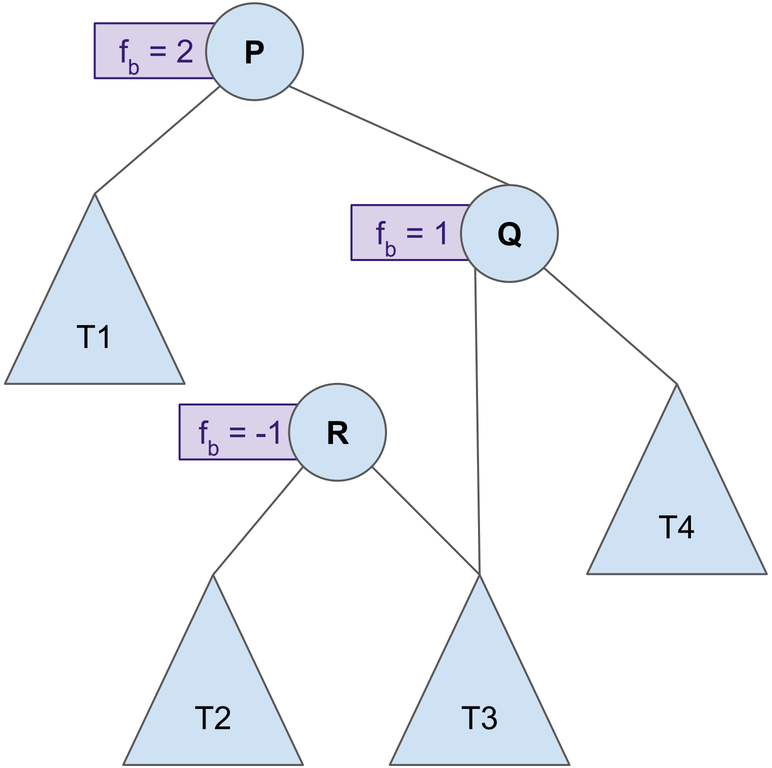
```
Entrada: nó P
1.  rotSimplesDir(q)
  1.1.  r = q->esq
  1.2.  q->esq = r->dir
  1.3.  r->dir = q
2.  rotSimplesEsq(p)
  2.1.  r = p->dir
  2.2.  p->dir = r->esq
  2.3.  r->esq = p
```



Rotações

ROTAÇÃO DUPLA À ESQUERDA

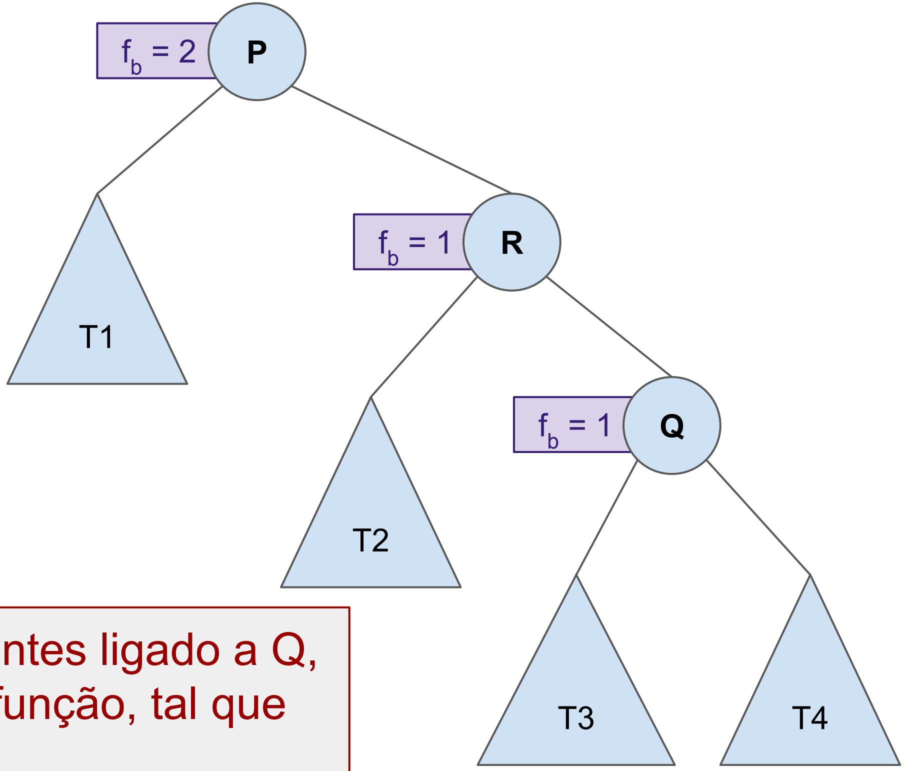
```
Entrada: nó P
1.   rotSimplesDir(q)
   1.1.   r = q->esq
   1.2.   q->esq = r->dir
   1.3.   r->dir = q
2.   rotSimplesEsq(p)
   2.1.   r = p->dir
   2.2.   p->dir = r->esq
   2.3.   r->esq = p
```



Rotações

ROTAÇÃO DUPLA À ESQUERDA

```
Entrada: nó P
1.   rotSimplesDir(q)
  1.1.   r = q->esq
  1.2.   q->esq = r->dir
  1.3.   r->dir = q
2.   rotSimplesEsq(p)
  2.1.   r = p->dir
  2.2.   p->dir = r->esq
  2.3.   r->esq = p
```

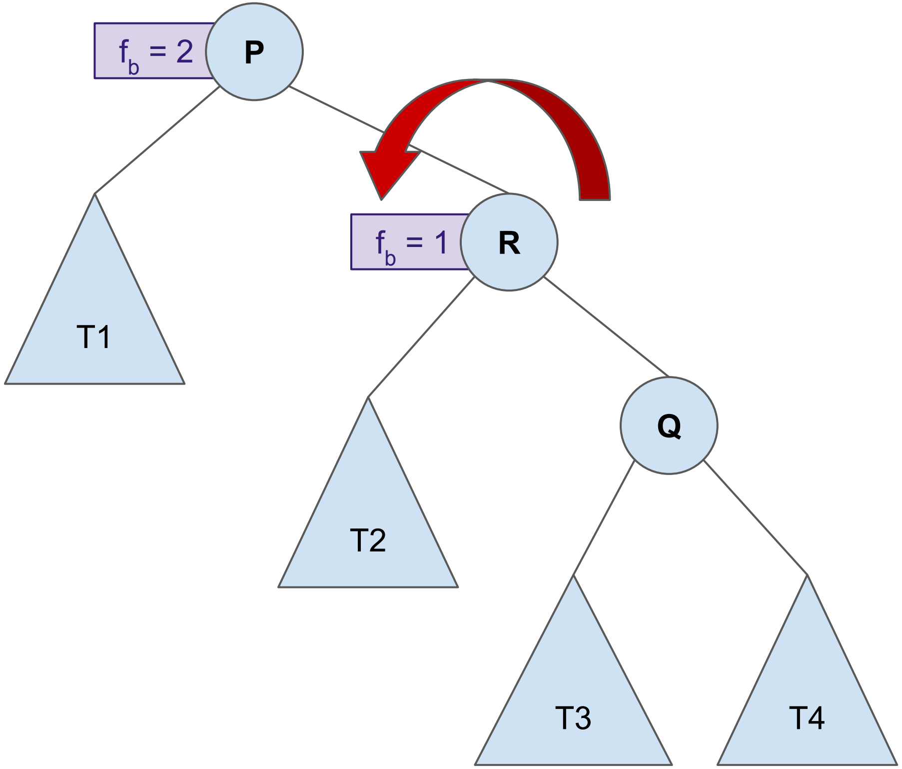


Obs.: o ajuste do ponteiro direito de P, antes ligado a Q, deve ser feito retornando R ao fim da função, tal que P->dir = R

Rotações

ROTAÇÃO DUPLA À ESQUERDA

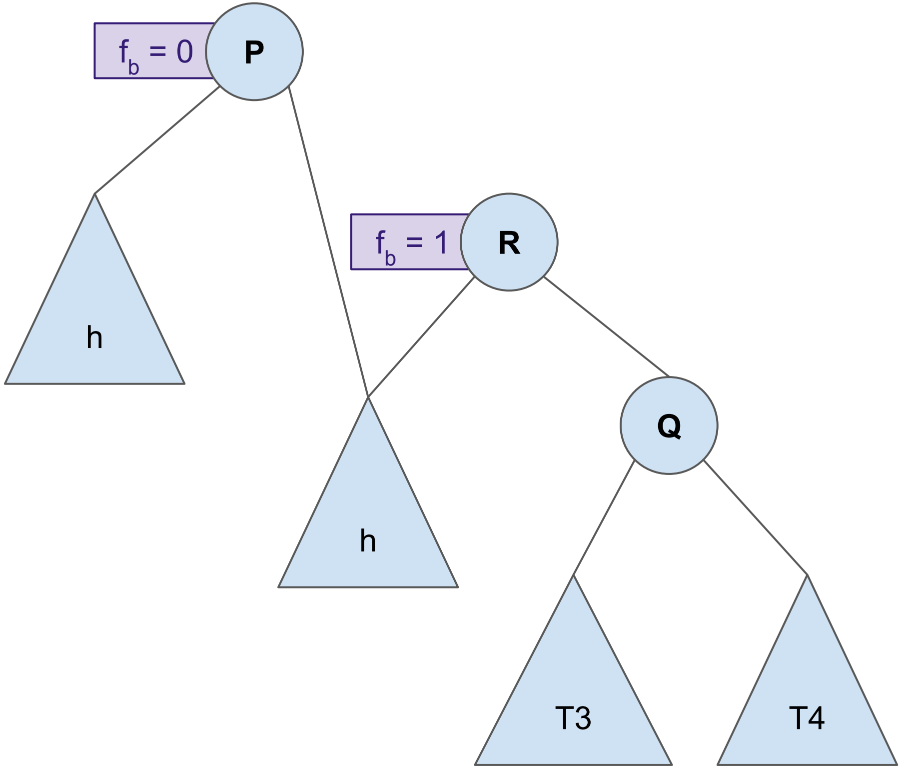
```
Entrada: nó P
1.   rotSimplesDir(q)
   1.1.   r = q->esq
   1.2.   q->esq = r->dir
   1.3.   r->dir = q
2.   rotSimplesEsq(p)
   2.1.   r = p->dir
   2.2.   p->dir = r->esq
   2.3.   r->esq = p
```



Rotações

ROTAÇÃO DUPLA À ESQUERDA

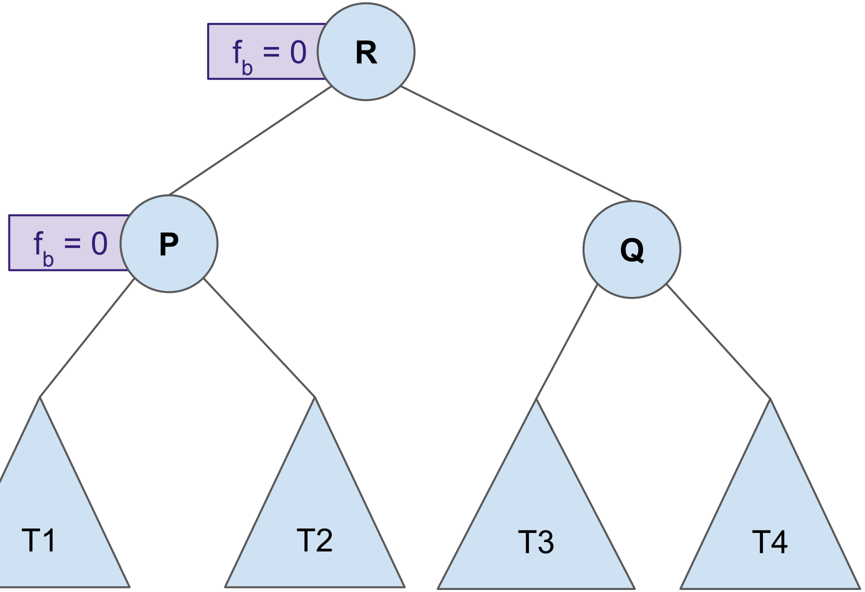
```
Entrada: nó P
1.   rotSimplesDir(q)
   1.1.   r = q->esq
   1.2.   q->esq = r->dir
   1.3.   r->dir = q
2.   rotSimplesEsq(p)
   2.1.   r = p->dir
   2.2.   p->dir = r->esq
   2.3.   r->esq = p
```



Rotações

ROTAÇÃO DUPLA À ESQUERDA

```
Entrada: nó P
1.   rotSimplesDir(q)
   1.1.   r = q->esq
   1.2.   q->esq = r->dir
   1.3.   r->dir = q
2.   rotSimplesEsq(p)
   2.1.   r = p->dir
   2.2.   p->dir = r->esq
   2.3.   r->esq = p
```



Obs.: o ajuste do ponteiro do pai de R, antes ligado a P, deve ser feito retornando R ao fim da função, tal que pai->(esq | dir) = R

Rotações

- Como saber qual rotação aplicar?
  - Analisando os fatores de balanceamento dos nós envolvidos
    - Rotação simples à esquerda
      - f(P) = +2
      - f(Q) = +1 ou 0
    - Rotação simples à direita
      - f(P) = -2
      - f(Q) = -1 ou 0
    - Rotação dupla à esquerda
      - f(P) = +2
      - f(Q) = -1
    - Rotação dupla à direita
      - f(P) = -2
      - f(Q) = +1

UMA IMPLEMENTAÇÃO EM C++

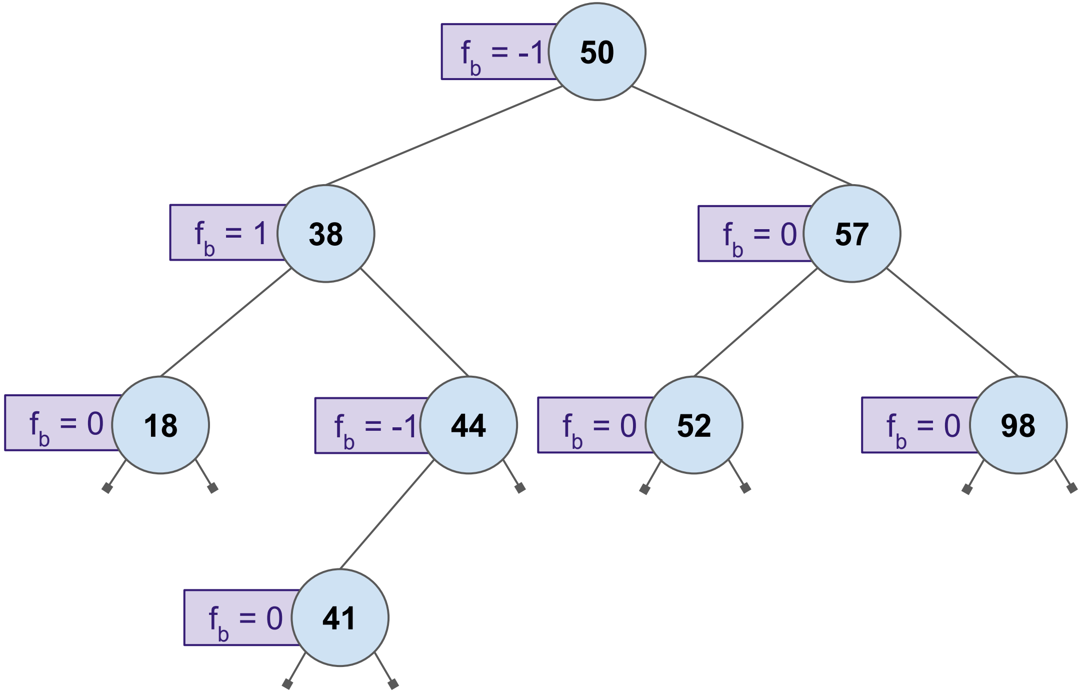
```
NoAVL* rotSimplesEsq(NoAVL* p)
{
    NoAVL *q = p->dir;
    p->dir = q->esq;
    q->esq = p;
    return q;
}
```

Inserção em AVL

Ponteiro para NULL

Inserir 40

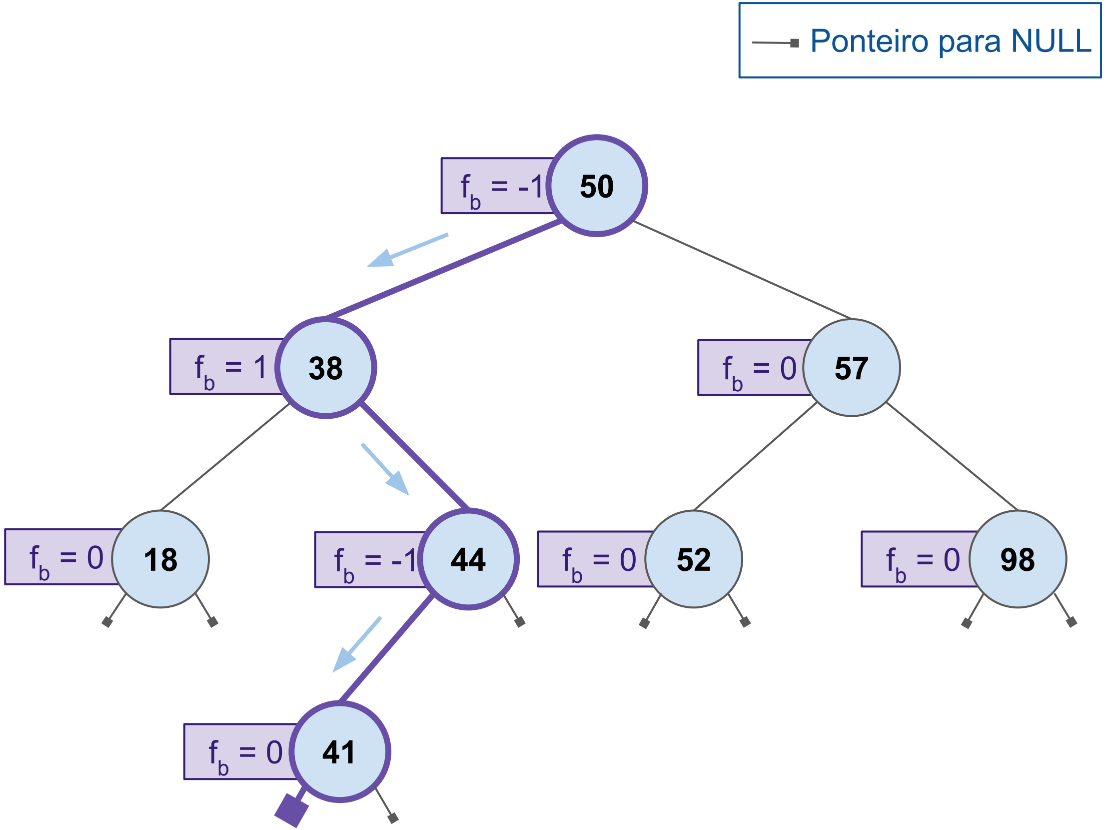
Procedimento de inserção é o mesmo da árvore binária de busca



Inserção em AVL

Inserir 40

1. Encontra a posição de inserção

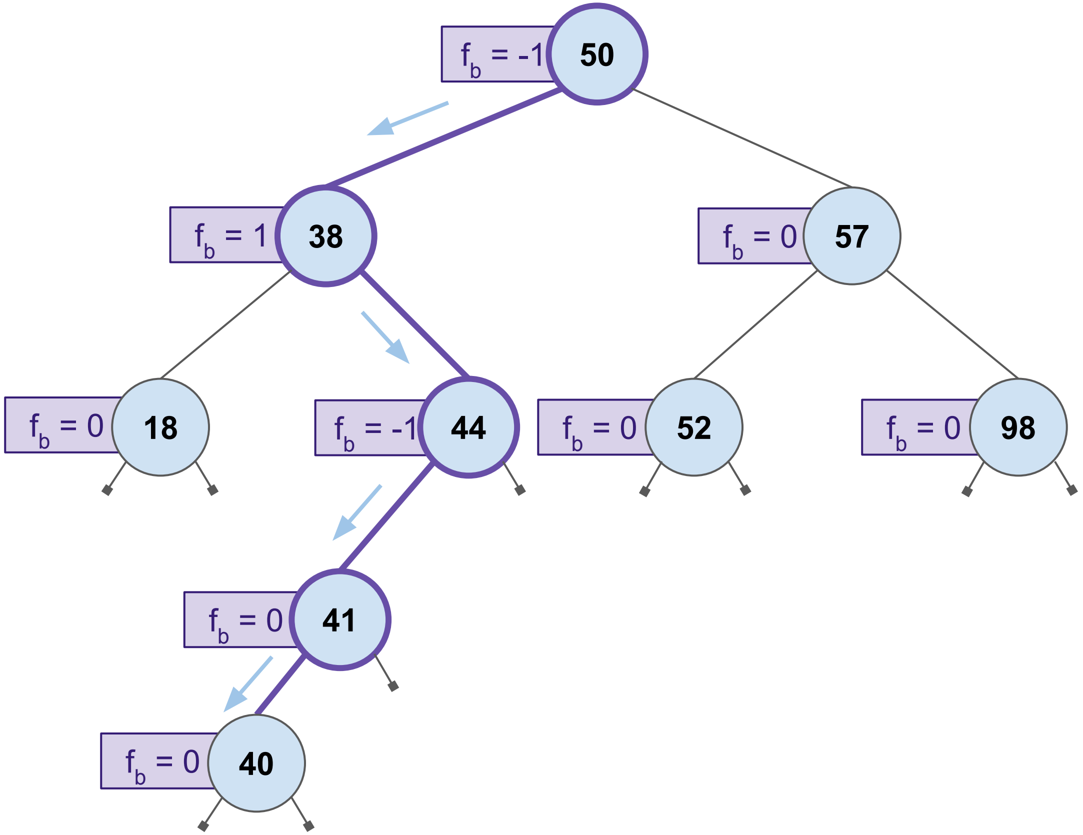


Inserção em AVL

Ponteiro para NULL

Inserir 40

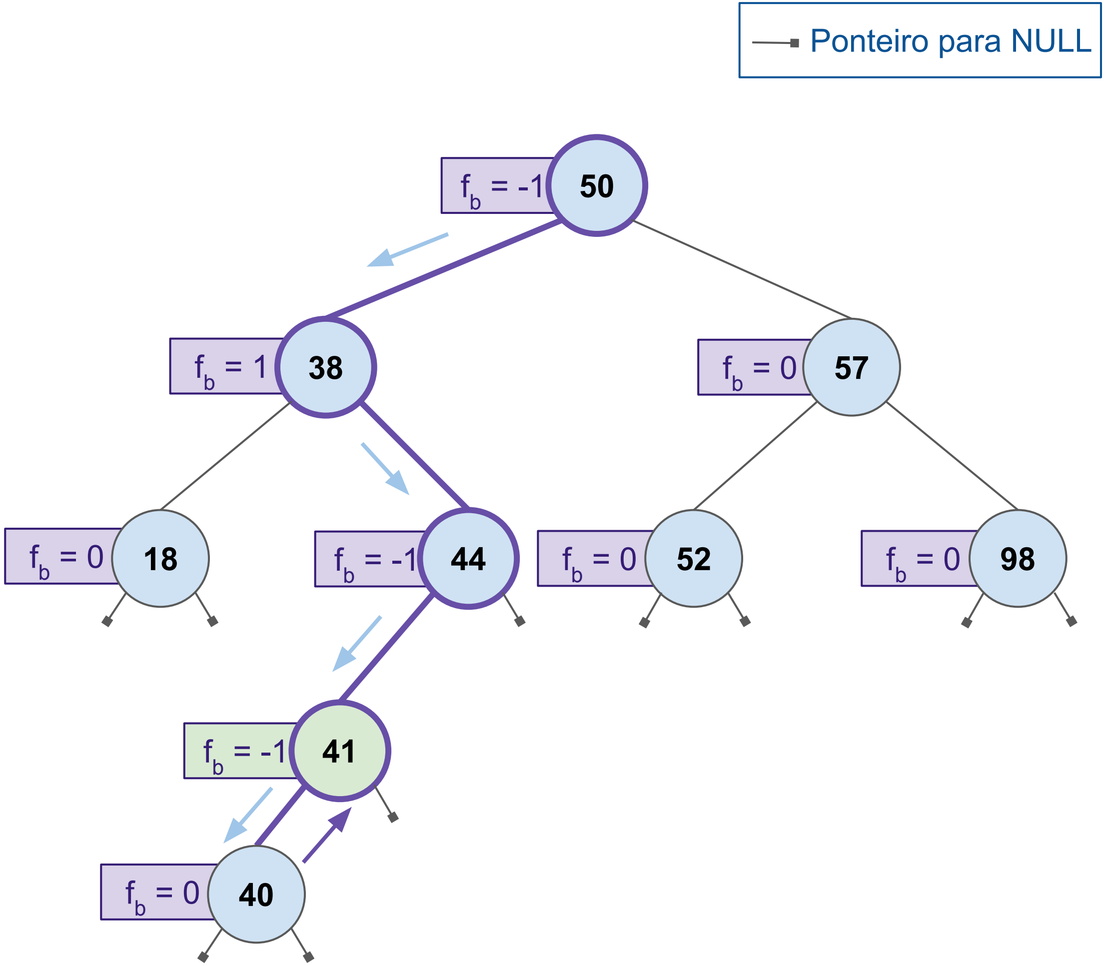
1. Encontra a posição de inserção
2. Insere novo nó



Inserção em AVL

⟹ **Inserir 40**

```
1.   Encontra a posição
     de inserção
2.   Insere novo nó
3.   Retorna atualizando
     os fatores
```

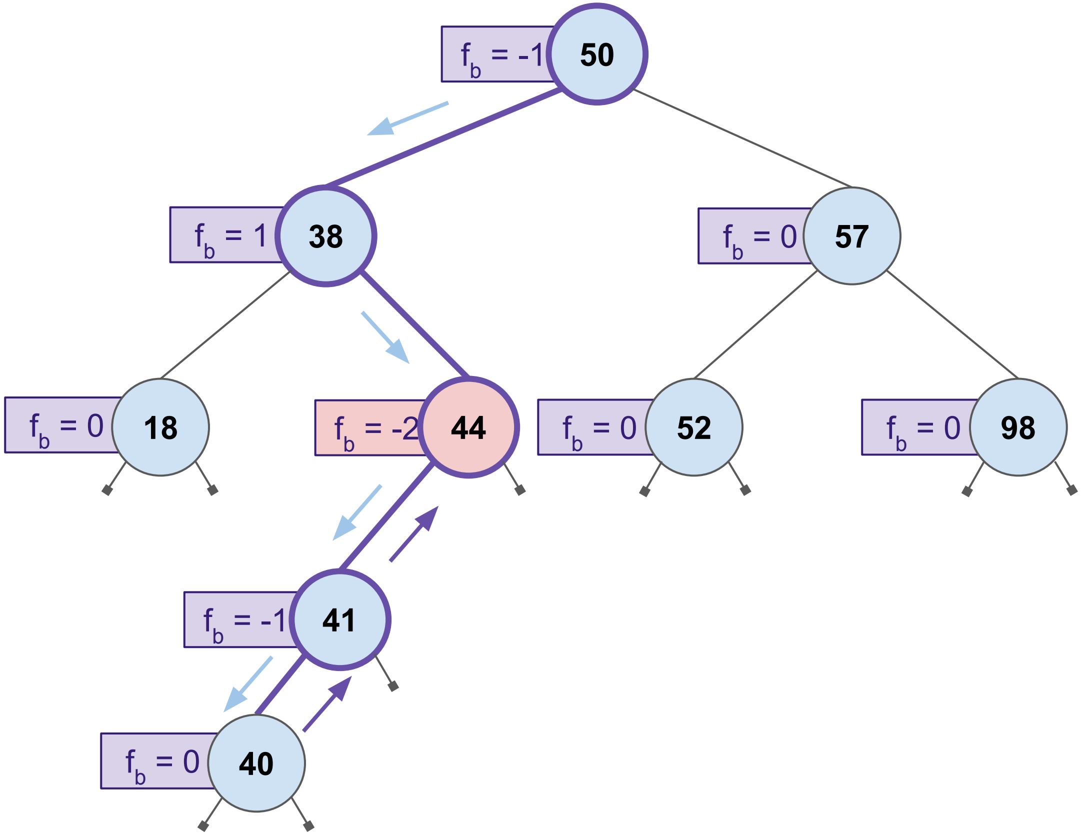


Inserção em AVL

Ponteiro para NULL

Inserir 40

1. Encontra a posição de inserção
2. Insere novo nó
3. Retorna atualizando os fatores
4. Nó desbalanceado detectado



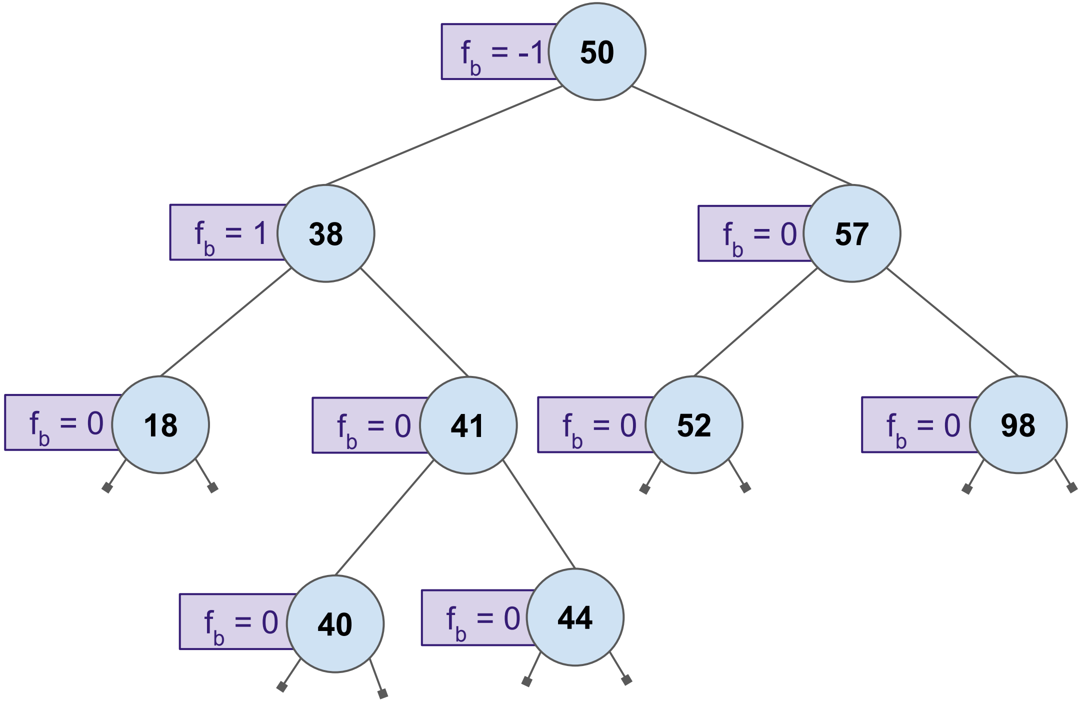
Inserção em AVL

Ponteiro para NULL

Inserir 40

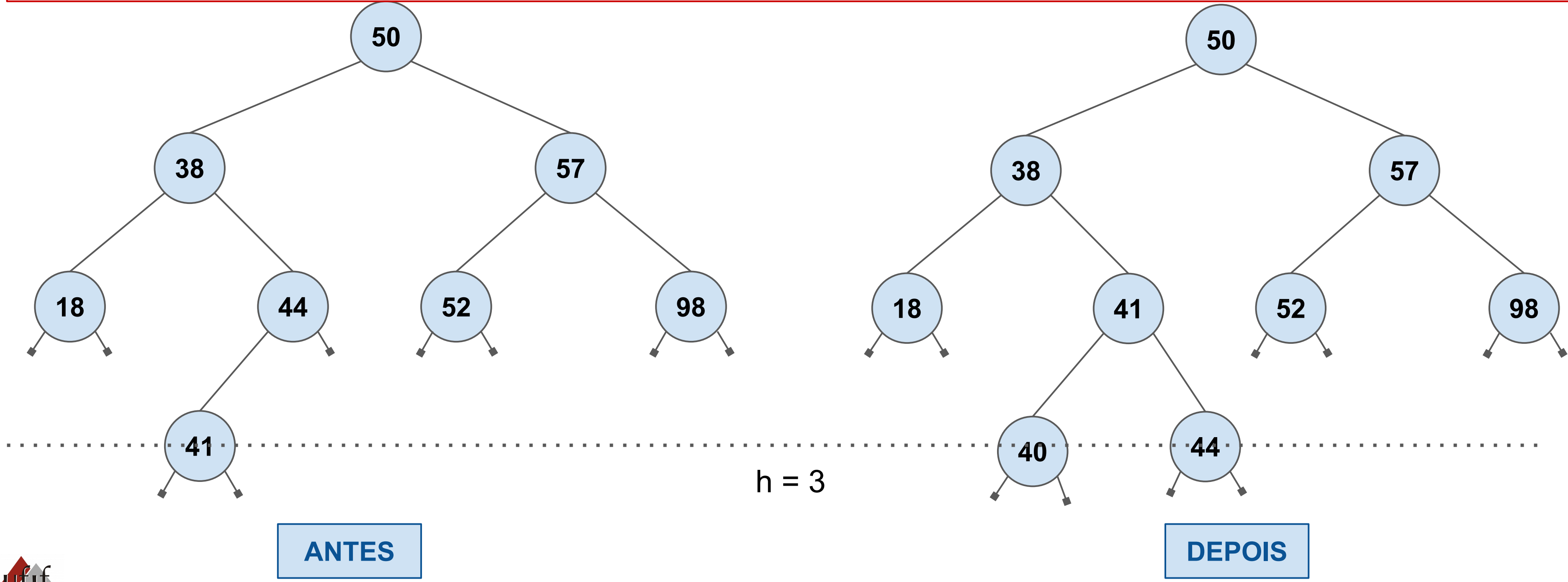
1. Encontra a posição de inserção
2. Insere novo nó
3. Retorna atualizando os fatores
4. Nó desbalanceado detectado
5. Aplica rotação

O processo se encerra após a rotação



Inserção em AVL

Note que não é necessário atualizar os demais fatores de balanceamento após a rotação, pois a altura da árvore antes da inserção e após o balanceamento é a mesma

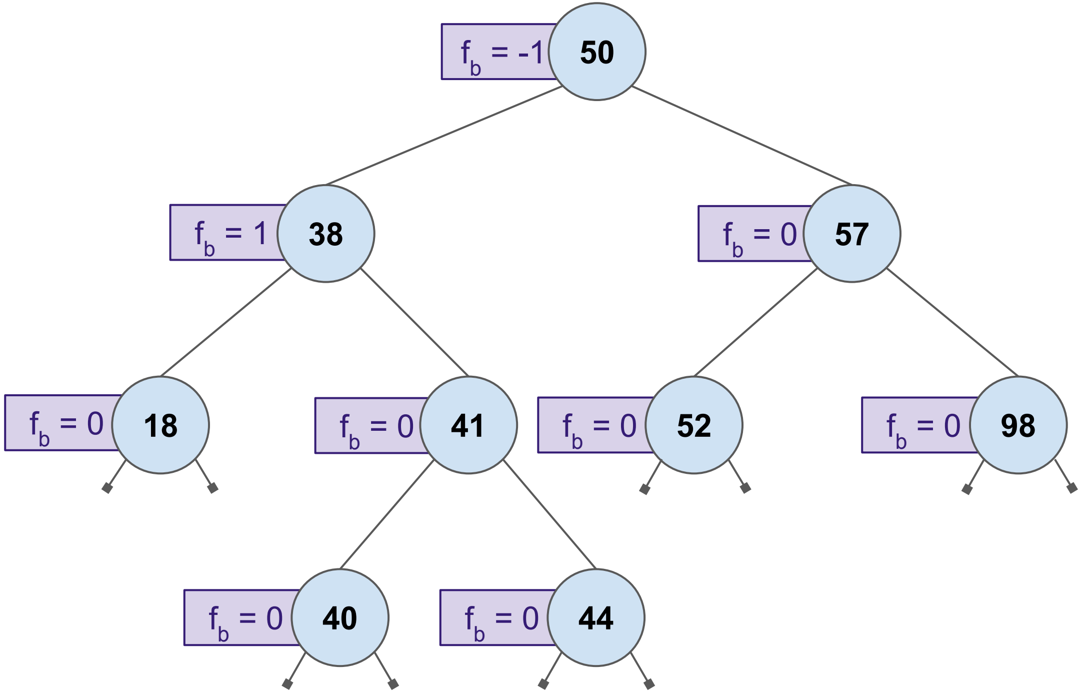


Remoção em AVL

Ponteiro para NULL

Remover 18

Procedimento de remoção é o mesmo da árvore binária de busca

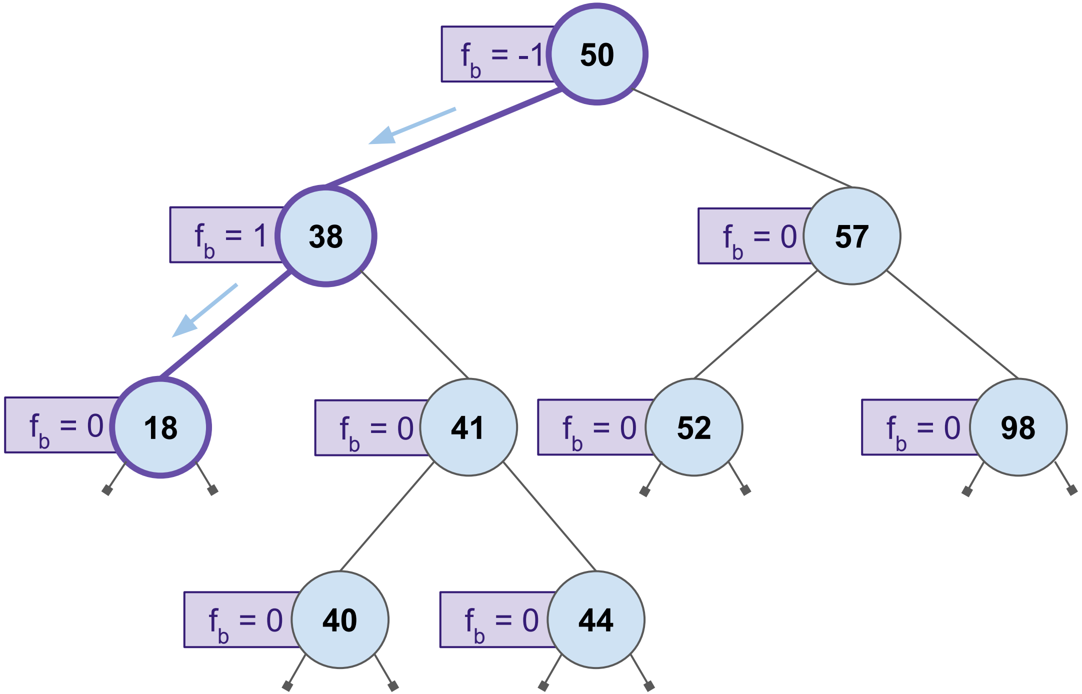


Remoção em AVL

Ponteiro para NULL

Remover 18

1. Encontra a posição de remoção

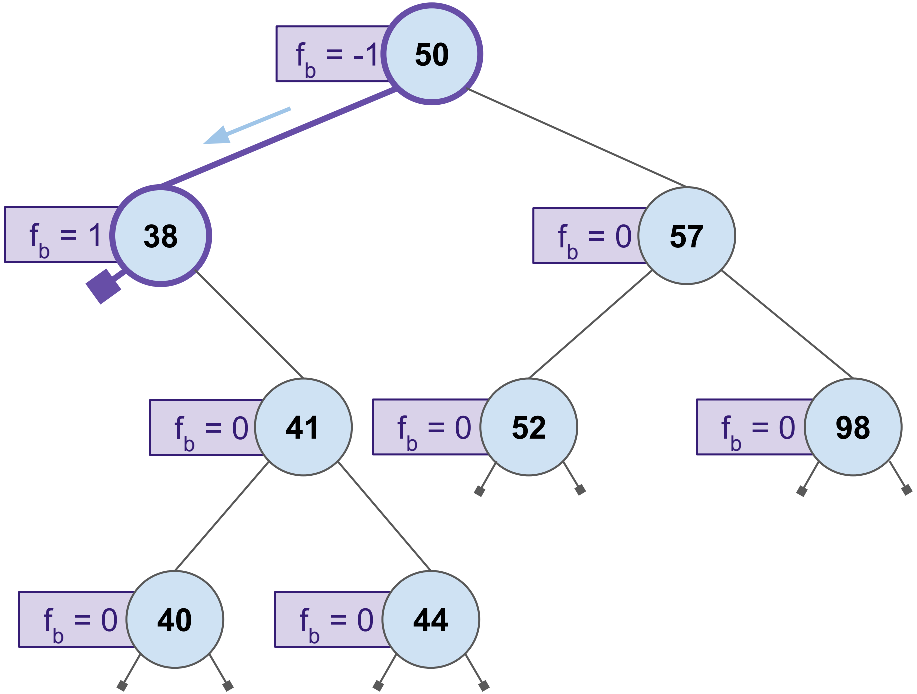


Remoção em AVL

Ponteiro para NULL

Remover 18

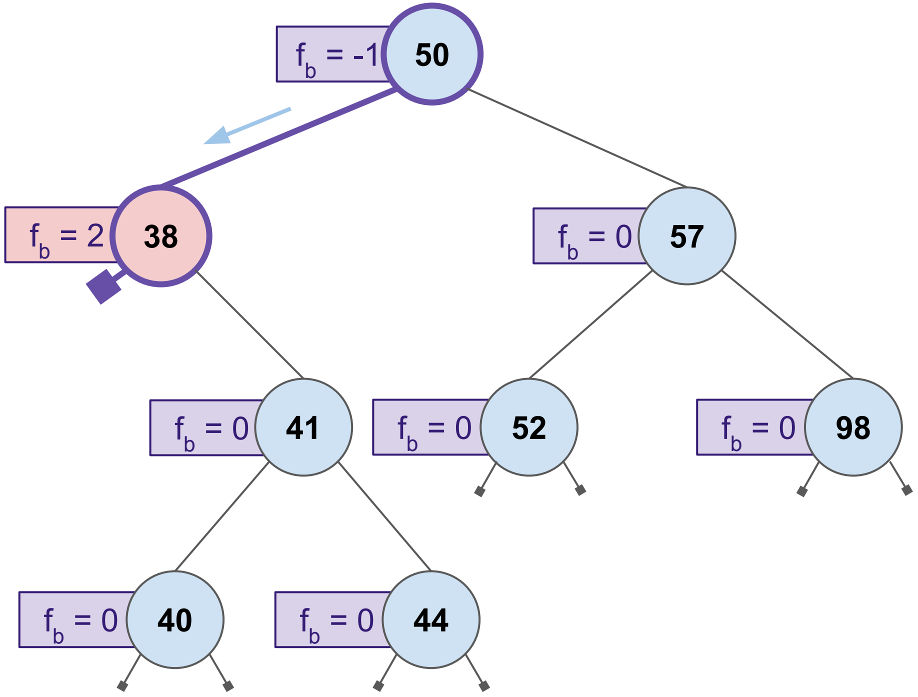
1. Encontra a posição de remoção
2. Remove nó (nó folha)



Remoção em AVL

Remover 18

1. Encontra a posição de remoção
2. Remove nó (caso nó folha)
3. Retorna atualizando os fatores
4. Nó desbalanceado detectado

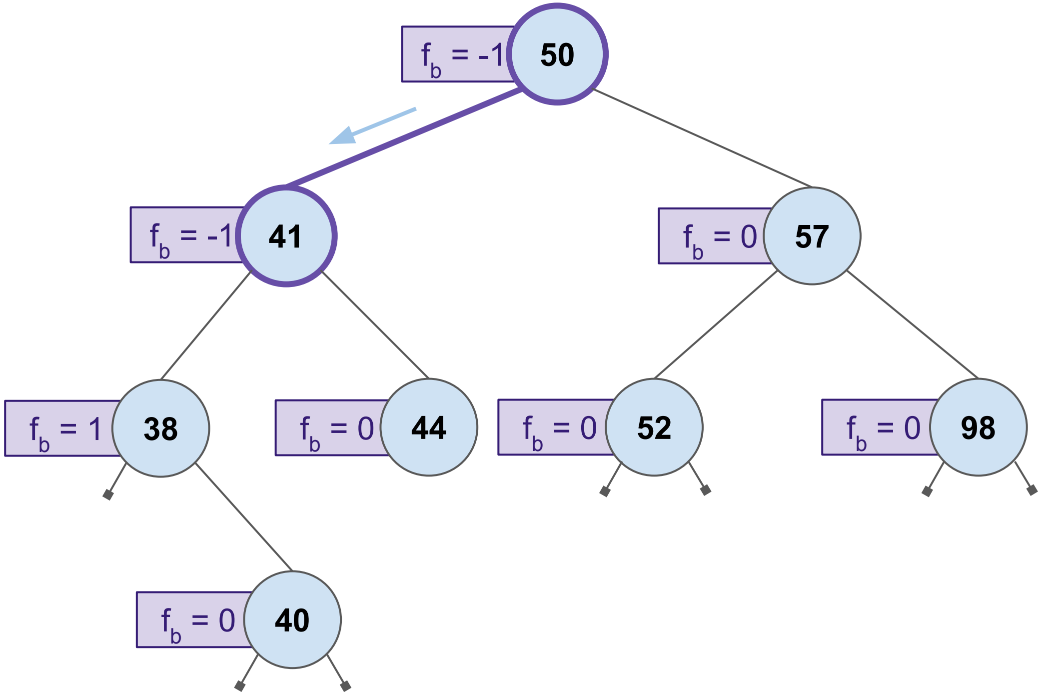


Remoção em AVL

Ponteiro para NULL

Remover 18

1. Encontra a posição de remoção
2. Remove nó (caso nó folha)
3. Retorna atualizando os fatores
4. **Nó desbalanceado detectado**
5. Aplica rotação

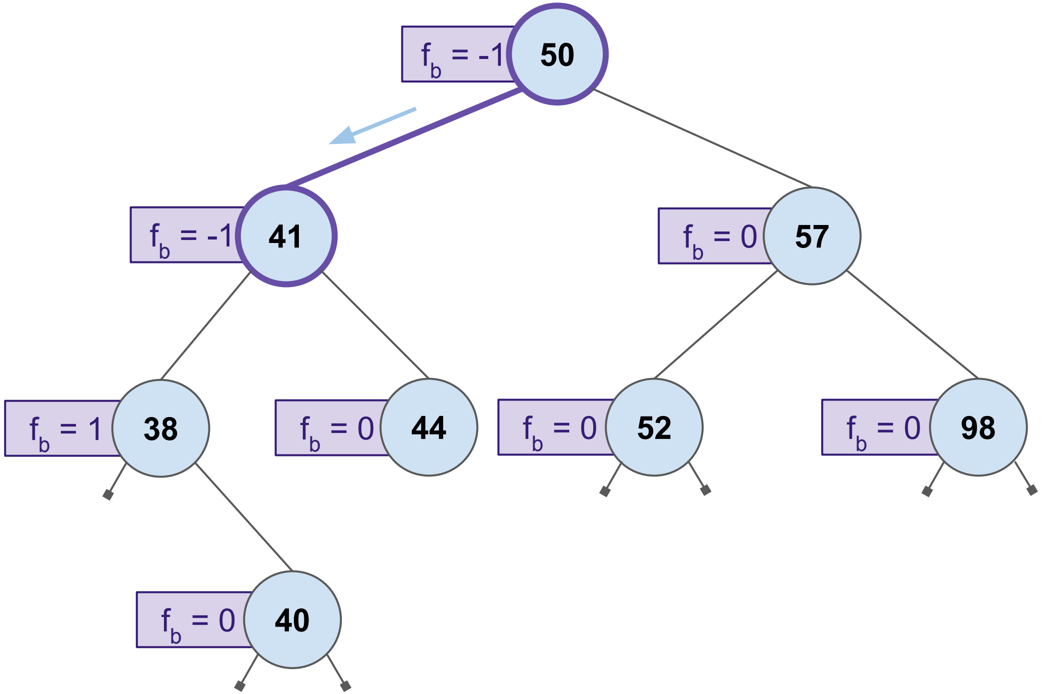


Remoção em AVL

Ponteiro para NULL

Remover 18

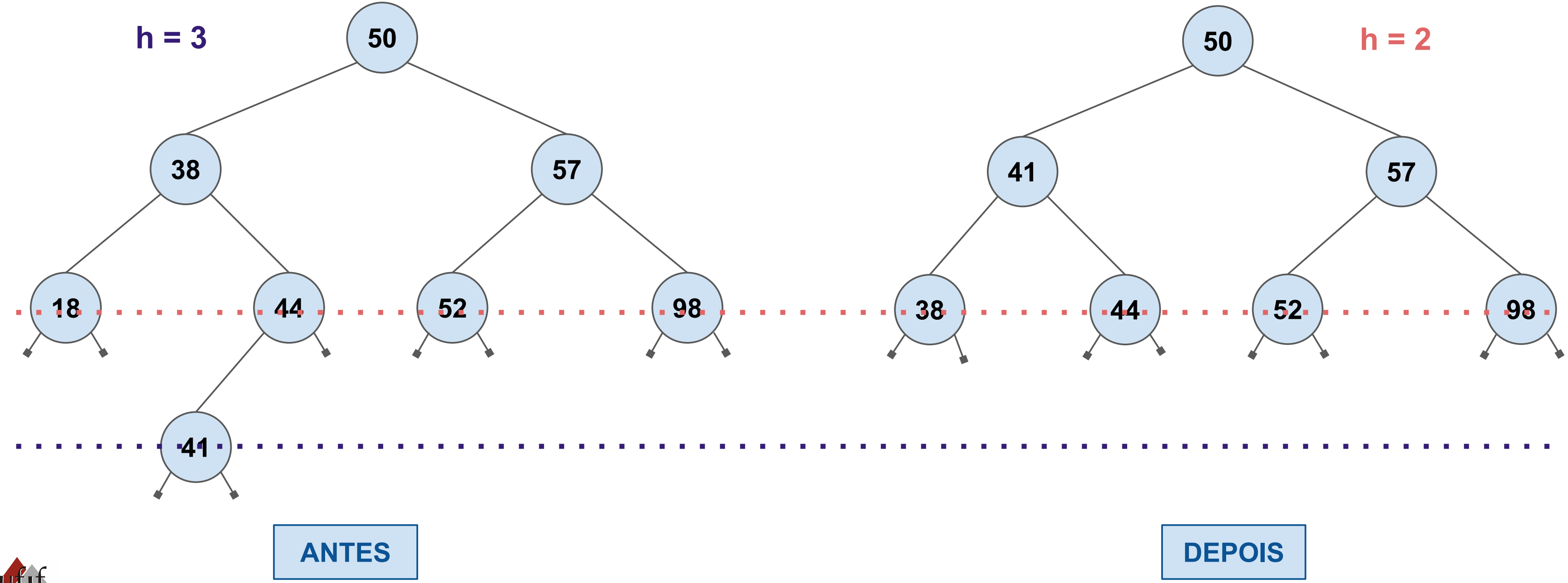
1. Encontra a posição de remoção
2. Remove nó (caso nó folha)
3. Retorna atualizando os fatores
4. Nó desbalanceado detectado
5. Aplica rotação



Na remoção, no entanto, o rebalanceamento **não termina** após a rotação

Remoção em AVL

Suponha que quiséssemos remover o 18 na árvore original, antes da inserção do 40. Note que a altura da árvore após a rotação diminui, o que impacta os fatores de balanceamento de todos os nós no caminho de 18 até a raiz.



Árvore AVL

- Assim, a inserção em AVL requer no máximo 1 rotação (simples ou dupla) para balancear a árvore e até O(logn) atualizações de fatores
- Já a remoção pode, no pior caso, gerar O(logn) rotações
- Aplicável em situações em que poucas inserções e remoções são esperadas
  - Árvores vermelho-preto são mais eficientes em inserções e remoções

Exercício

Exercício 1:

Monte a árvore AVL (passo-a-passo) para as seguintes inserções de chaves 41, 38, 31, 12, 19, 8, 27, 49 (nesta ordem), indicando a cada passo qual elemento foi inserido e qual rotação foi realizada.

Exercício 2:

Construa a árvore AVL resultante da inserção das seguintes chaves na ordem especificada:

- 12, 70, 80, 43, 41, 60, 82, 90, 99, 11, 64

Referências

- SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. Estruturas de Dados e Seus Algoritmos, cap. 5. LTC Editora, 1994.
- DROZDEK, Adam. Data Structures and Algorithms in C++, Fourth Edition, cap. 10. Cengage Learning, 2013.
- SOUZA, Jairo F. Notas de aula de Estrutura de Dados II. 2016. Disponível em: http://www.ufjf.br/jairo_souza/ensino/material/ed2/